IBM 1130 Computing System User's Guide

This manual, covering a wide range of subjects that are of interest to 1130 customer personnel, is designed for insertion in a workbook along with user-generated materials. It deals with the steps to be considered in any successful installation program: preinstallation planning, documenting current applications, design of new applications, conversion, program development, testing, and program documentation.

Additional topics discussed include the 1130 system, the 1130 Monitor, Job Management, Disk Management, Core Management, File Organization, Disk Data Storage, FORTRAN and the Commercial Subroutine Package, Sorting and System Evaluation - Performance.

It is suggested that the User's Guide be placed in a binder and that dividers be inserted before the various sections. The resulting workbook becomes the single major source of installation guidance when you include your own data processing policies, standards, and control forms.
READER'S GUIDE

INTRODUCTION

This section is intended as a guide to help you get the most out of this manual. Because of the magnitude of the manual and the differing needs of various readers, such a guide, or "road map", is particularly important.

For purposes of the guide, readers will be divided into three groups:

1. Top management, who want an overview of the system.
2. Data processing management, who have direct responsibility for the installation and management of the 1130 System.
3. Programmers and systems analysts, who will actually set up the system, determine the methods to be used, and/or code the programs.

Groups 2 and 3 are subdivided into those concerned with "pure" scientific applications, and those working in a commercial or mixed scientific-commercial atmosphere.

Figure 01.1 shows a general outline of the manual and suggests which sections should be read by each group. However, the top manager who wants a more detailed view of the 1130 will find much of the data processing management material to be relevant; the data processing manager may want to read more than Figure 01.1 indicates; etc.

The effectiveness of this Guide depends entirely on the responsible manager in your installation. The Guide contains possible paths to a successful installation. Since the installation of data processing equipment is a disciplined venture that involves decisions concerning the selection of the best paths, your management's responsibility is clearly delineated. This responsibility began with the creation of realistic objectives. Control is exercised through timely reviews in which progress is related to checkpoints and corrective action is undertaken.

A WORD TO TOP MANAGEMENT

Within the last several years, your company may have increased its plant capacity to meet growing needs. Before this new resource became fully operational, though, many things had to be done. Management was chosen, an organization chart drawn up, a plan for startup formulated, a date picked for the start of production, etc.

Just recently you may have added a new product or service. The introduction of this product or service involved many considerations. Its need was studied, its function determined, an announcement date selected, etc.

In both cases, management:
- Defined its objectives
- Made a plan
- Established checkpoints
- Assigned responsibilities

Timely reviews determined whether your plans were being followed, your objectives met, etc. On the basis of these reviews, modifications and adjustments were made to ensure that the operation was a success.

Now you are adding another resource to your organization — an IBM 1130 Computing System. As before, there are many things that you, as management, must do if your 1130 installation is to meet its planned objectives.

Should the installation of a new data processing system be any less subject to management control than a new plant, or a new product? The answer is no. Data processing capability is a resource, just like the new plant or new product. In fact, a data processing system is a unique type of resource; it is one that extends management's ability to control other resources.

Your 1130 system may be used to maintain a personnel skills inventory or to schedule plant operations. It may be assigned to keep a close watch on cash flow or to determine reorder points for your inventory. In each case, data processing is a resource being used to control other resources.

In this light, the IBM 1130 Computing System that you are about to install should take on an added importance. Objectives, checkpoints, and the mechanics for review should be established for this resource, just as for any other resource available to you.

The 1130 Computing System, through its stored-program power and random access disk capability, embodies a new technology. The maximum value will be derived from this technology only if the system is oriented toward your objectives and its installation is closely monitored to see that those objectives are achieved. It is through this type of involvement that the philosophies and policies of a manager can be manifested.

The 1130 User's Guide has been designed with these thoughts in mind. First, it deals with all the considerations that lead to a successful installation. Second, it is so organized as to lend itself to the control and review process. The cornerstone of
<table>
<thead>
<tr>
<th>THE TOPIC:</th>
<th>THE READER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top Mgmt (Commercial-Scientific)</td>
</tr>
<tr>
<td>Preinstallation Planning</td>
<td>✓</td>
</tr>
<tr>
<td>Documenting Current Applications</td>
<td>Introduction</td>
</tr>
<tr>
<td>Preliminary Questions &amp; Answers</td>
<td>Cards vs Disk Files</td>
</tr>
<tr>
<td></td>
<td>Safeguarding Data</td>
</tr>
<tr>
<td>Application Design</td>
<td>Accounting Controls</td>
</tr>
<tr>
<td></td>
<td>Forms Design</td>
</tr>
<tr>
<td></td>
<td>Card Design</td>
</tr>
<tr>
<td></td>
<td>Disk Design</td>
</tr>
<tr>
<td></td>
<td>Program Development</td>
</tr>
<tr>
<td></td>
<td>Testing Effectively</td>
</tr>
<tr>
<td></td>
<td>Documentation</td>
</tr>
<tr>
<td></td>
<td>Conversion</td>
</tr>
<tr>
<td></td>
<td>The 1130 System</td>
</tr>
<tr>
<td></td>
<td>The 1130 Monitor</td>
</tr>
<tr>
<td></td>
<td>Job Management</td>
</tr>
<tr>
<td></td>
<td>Disk Management</td>
</tr>
<tr>
<td></td>
<td>Increasing Space</td>
</tr>
<tr>
<td></td>
<td>Disk Util. Prog.</td>
</tr>
<tr>
<td>Core Storage Management</td>
<td>✓</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>Arithmetic</td>
</tr>
<tr>
<td></td>
<td>Overlapped I/O</td>
</tr>
<tr>
<td></td>
<td>Character Handling</td>
</tr>
<tr>
<td></td>
<td>Core Saving Tips</td>
</tr>
<tr>
<td></td>
<td>Timing</td>
</tr>
<tr>
<td>Sorting</td>
<td>Introduction</td>
</tr>
<tr>
<td>Use of the Disk for Data Storage</td>
<td>Introduction</td>
</tr>
<tr>
<td>Disk Organization and Processing</td>
<td>✓</td>
</tr>
<tr>
<td>Improving Your System – Performance</td>
<td>Introduction</td>
</tr>
</tbody>
</table>

✓  Read this section
*  May be skipped if you don’t have, or are not considering using, the disk for data storage

Figure 01.1
this organization is an Installation Activity Schedule, which highlights all the events leading to a successful installation. This is fully described in Section 05.

This Guide should become a working document in your organization. Although the experience and specific needs of each organization vary considerably, all the events apply to some extent in every installation.

A WORD TO DATA PROCESSING MANAGEMENT

In addition to the comments directed toward top management, several thoughts apply here.

You are the men in the middle — between top management and the programmer/analyst. For this reason the sections checked for your attention are those concerned with how to do things the "right" way; how to avoid potential pitfalls; how to get the most out of your 1130 system; etc.

A WORD TO THE PROGRAMMER/ANALYST

As Figure 01.1 shows, this manual is directed primarily toward you; you should read its entire contents. This is especially true for those of you who are working in a commercial, or mixed, environment.

However, the distinction between a commercial, or mixed, environment and a "pure" scientific environment is very tenuous. More and more, users who once considered themselves "pure" scientific find their applications taking on aspects of the traditional commercial job — large data files are developed, input and output formats become more critical, alphabetic codes and data are encountered.

Actually, the subjects checked for the "pure" scientific reader represent a bare minimum. Anyone who is or expects to be in a mixed environment should read the entire manual.

SUMMARY OF THE USER'S GUIDE

The Installation Phase

The following listing of the material in this Guide reflects the major grouping of installation events and should provide an indication of the Guide's comprehensive nature. Comments have been added to each listed item to relate the manner in which that subject matter may be used.

- Preinstallation Planning — provides a proven method of scheduling and reviewing installation activities, specifically tailored to the 1130 user, and illustrates the points where management review is most essential.

- Documenting Current Applications — concerns the control and techniques that can be applied to the documentation of existing procedures. Distinction is made between manual operations and those already mechanized.

- Some Preliminary Questions and Answers Regarding Data Storage — considers the pros and cons of using either cards or disk for data storage. Also considers protecting your data — why and how it should be protected.

- 1130 Application Design — includes card and form design, record layouts, and flowcharts. The elements of application design are made clear through "live" illustrations, which are used throughout. This section also aids in the selection of the right job-oriented programming language and thus contributes to the effectiveness of the whole installation effort.

- Program Development — devotes itself to converting designs for 1130 applications into tested, debugged machine programs. The application discussed throughout the Guide is provided to serve as a teaching aid and time saver for the programmer. Programming hints and aids are also provided.

- Testing Effectively — shows the methods an installation should use in testing individual programs and complete systems.

- Program Documentation — shows how a good set of working documents, which a computer installation must develop, can be created during the development phases.

- Conversion — outlines the procedures required to perform the cutover from your present system to the 1130.

The Operations Phase

This portion of the Guide contains several sections of interest to users who have completed the installation phase:

- 1130 Computing System — contains a comprehensive description of the 1130 System and a brief description of each component.

- 1130 Disk Monitor System — discusses the 1130 Monitor in general and leads into the more detailed material of the next three sections.

- Job Management — covers those features of the Monitor that help you manage the job, or unit of work.

- Disk Management — describes the layout of disk storage, how you may use it, and how to get the most out of it.
- Core Storage Management — outlines the facilities the Monitor gives you to manage core storage with the LOCAL, SOCAL and LINK overlay systems.

- FORTRAN — General and Commercial — covers many aspects of FORTRAN that are of interest to all users, but with special emphasis on the needs of commercial programmers. Use of the Commercial Subroutine Package, arithmetic considerations, and core-saving tips are among the major topics covered.

- Sorting with Your 1130 — describes the sorting process and some alternate approaches.

- Use of the Disk for Data Storage — describes the way data is situated on the disk, and stresses efficiency.

- Disk Data Files — Organization and Processing — continues the previous topic, discussing the various file organization techniques and how the processing sequence affects the choice of organization.

- Improving Your System — Performance — covers performance and how it is affected by (1) the Monitor, (2) the programmer, and (3) the 1130 itself. Three case studies are presented to illustrate various approaches to improving throughput rates and run times.
### CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Subsections</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
</tr>
</tbody>
</table>

#### Section 01: Reader's Guide

#### Section 02: Table of Contents

#### Section 05: Preinstallation Planning

- **Section Contents** 05.00.00
- **Introduction** 05.01.00
- **General Planning** 05.10.00
- **Application and Conversion Planning** 05.20.00
- **Programming Planning** 05.30.00

#### Section 10: Documenting Current Applications

- **Section Contents** 10.00.00
- **Introduction** 10.01.00
- **Documentation of Manual Systems** 10.10.00
- **Documentation of Punched Card Systems** 10.20.00
- **Accounting Controls** 10.30.00
- **Survey Questionnaires** 10.40.00
  - **Billing** 10.40.10
  - **Accounts Receivable** 10.40.20
  - **Sales Analysis** 10.40.30
  - **Inventory** 10.40.40
  - **Accounts Payable** 10.40.50
  - **Payroll** 10.40.60
- **Manual System Documentation Example—Payroll** 10.50.00
- **Introduction** 10.50.01
- **Job Description** 10.50.02
- **Survey Form** 10.50.03
- **Sample Documents** 10.50.04
- **Systems Flowchart** 10.50.05

#### Section 15: Some Preliminary Questions and Answers Regarding Data Storage

- **Section Contents** 15.00.00
- **Introduction** 15.01.00
- **Data -- On Disk or Cards?** 15.10.00
- **General Considerations** 15.10.01
- **Flexibility in Order of Processing** 15.10.02
- **Jobs Involving More than One File** 15.10.03
- **Frequency of Changes to Your File** 15.10.04
- **Need for Inquiry into Your File** 15.10.05
- **Size of Your Data File** 15.10.06
- **Your Backup Requirements** 15.10.07
- **Record Size** 15.10.08
- **Other Considerations** 15.10.09
- **Summary** 15.10.10

**How to Safeguard Your Disk Data**

- **Files** 15.20.00
  - **Introduction** 15.20.01
  - **Know Your Data** 15.20.02
  - **Know What Can Happen to Your Data** 15.20.03
  - **Design an Accident-Insensitive System** 15.20.30
  - **Detect Errors Before They Do Damage** 15.20.40
  - **Plan Modest-Size, Modular Programs** 15.20.50
  - **Always Back Up Your Disk Files with a Duplicate Copy** 15.20.60
  - **Provide Tested and Documented Recovery Procedures** 15.20.70

#### Section 20: 1130 Application Design

- **Section Contents** 20.00.00
- **Introduction** 20.01.00
- **Accounting Controls** 20.10.00
  - **Review of Accounting Control Principles** 20.10.01
  - **More Specific Suggestions for Document and Accounting Controls** 20.10.02
- **Form Design** 20.20.00
  - **1130 Considerations** 20.20.01
  - **Form Design Principles** 20.20.02
- **Card Design** 20.30.00
  - **1130 Considerations** 20.30.01
  - **Card Design Principles** 20.30.02
- **Design of Disk Data Files** 20.40.00
  - **Introduction** 20.40.01
  - **Data** 20.40.02
  - **Field Size** 20.40.03
  - **Data Sequence** 20.40.04
  - **File Organization** 20.40.05
  - **Record Format and Blocking** 20.40.06
  - **File Processing** 20.40.07
  - **File Control** 20.40.08
- **Payroll Example** 20.50.00
  - **Narrative** 20.50.01
  - **Card Forms and Console Keyboard Input** 20.50.02
  - **Console Printer and Line Printer** 20.50.03
  - **Disk Record Formats** 20.50.04
  - **System Flowchart** 20.50.05
  - **Language Selection** 20.50.06
  - **Introduction** 20.50.07
  - **Programming Languages** 20.50.08
<table>
<thead>
<tr>
<th>Section</th>
<th>Subsections</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>00</td>
<td>02</td>
</tr>
</tbody>
</table>

**Section 25: Program Development**

<table>
<thead>
<tr>
<th>Section Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>25.01.00</td>
</tr>
<tr>
<td>Programming and Documentation Standards</td>
<td>25.10.00</td>
</tr>
<tr>
<td>Program Change Authorization</td>
<td>25.20.00</td>
</tr>
<tr>
<td>Programming Aids</td>
<td>25.30.00</td>
</tr>
<tr>
<td>Documenting Variable Usage</td>
<td>25.30.10</td>
</tr>
<tr>
<td>Modular Programming</td>
<td>25.30.20</td>
</tr>
<tr>
<td>Programming Examples</td>
<td>25.40.00</td>
</tr>
<tr>
<td>Example 1: File Creation</td>
<td>25.40.10</td>
</tr>
<tr>
<td>Example 2: Add Name to the File</td>
<td>25.40.20</td>
</tr>
<tr>
<td>Example 3: Changes to the File</td>
<td>25.40.30</td>
</tr>
<tr>
<td>Example 4: Calculations and Payroll Register</td>
<td>25.40.40</td>
</tr>
<tr>
<td>Example 5: Check Writing</td>
<td>25.40.50</td>
</tr>
<tr>
<td>Example 6: Check Register</td>
<td>25.40.60</td>
</tr>
<tr>
<td>Example 7: 941 Report</td>
<td>25.40.70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 50: 1130 Disk Monitor System General</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>50.01.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 55: The Monitor - Job Management Section Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>55.01.00</td>
</tr>
<tr>
<td>Job and Subjob</td>
<td>55.10.00</td>
</tr>
<tr>
<td>Stacked Jobs or the Input Stream</td>
<td>55.20.00</td>
</tr>
<tr>
<td>Disk Cartridge ID Checking</td>
<td>55.30.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 60: The Monitor - Disk Management Section Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>60.01.00</td>
</tr>
<tr>
<td>Disk Storage Layout</td>
<td>60.10.00</td>
</tr>
<tr>
<td>Cylinder 0</td>
<td>60.10.01</td>
</tr>
<tr>
<td>IBM Systems Area</td>
<td>60.10.10</td>
</tr>
<tr>
<td>Working Storage (WS)</td>
<td>60.10.20</td>
</tr>
<tr>
<td>User Area (UA)</td>
<td>60.10.30</td>
</tr>
<tr>
<td>Fixed Area (FX)</td>
<td>60.10.40</td>
</tr>
<tr>
<td>Summary</td>
<td>60.10.50</td>
</tr>
<tr>
<td>Increasing the Amount of Space Available to the User</td>
<td>60.20.00</td>
</tr>
<tr>
<td>How Much Room Do I Have?</td>
<td>60.20.10</td>
</tr>
<tr>
<td>How Can I Make More Space Available?</td>
<td>60.20.20</td>
</tr>
<tr>
<td>Summary</td>
<td>60.20.30</td>
</tr>
<tr>
<td>The Disk Utility Program</td>
<td>60.30.00</td>
</tr>
<tr>
<td>Format of Material on the Disk</td>
<td>60.30.01</td>
</tr>
<tr>
<td>The Most Commonly Used DUP Functions</td>
<td>60.30.10</td>
</tr>
<tr>
<td>Special Options -- Multiple Disk 1130 Users</td>
<td>60.30.20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 65: The Monitor - Core Storage Management Section Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>65.01.00</td>
</tr>
<tr>
<td>General</td>
<td>90.30.01</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Productive Time That Cannot Be</td>
<td>90.30.10</td>
</tr>
<tr>
<td>Improved by Hardware Changes</td>
<td>90.30.20</td>
</tr>
<tr>
<td>Productive Time That Can Be</td>
<td></td>
</tr>
<tr>
<td>Improved by Hardware Changes</td>
<td></td>
</tr>
<tr>
<td>Nonproductive Time That Can Be</td>
<td>90.30.30</td>
</tr>
<tr>
<td>Reduced by Hardware Changes</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Some Case Studies of Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improvements</td>
</tr>
<tr>
<td>General</td>
</tr>
<tr>
<td>Case I</td>
</tr>
<tr>
<td>Case II</td>
</tr>
<tr>
<td>Case III</td>
</tr>
<tr>
<td>Summary</td>
</tr>
</tbody>
</table>
Section 05: PREINSTALLATION PLANNING

CONTENTS

Introduction ......................... 05.01.00
General Planning .................... 05.10.00
Application and Conversion Planning ... 05.20.00
Programming Planning .............. 05.30.00
INTRODUCTION

Now that your 1130 computing system is on order, what should you do next? When the 1130 computing system was proposed, mention was made that it could perform both scientific and commercial jobs. Some typical commercial jobs that may have been considered at that time are:

- Payroll (used as an example later in the manual) and labor distribution
- Accounts receivable
- Accounts payable
- Sales analysis
- Inventory control

Planning the use of the 1130 for specific applications such as the above leads to other questions that need answers. How will the personnel for your installation be selected? When will your applications be implemented on the 1130? How will this job of implementation be carried to completion? In other words, you need a plan to carry out the installation of this new system.

In answer to the first question, selection of personnel, your IBM representative can supply you with the Programmer's Aptitude Test, which will help you with some of the selection. (It may be that you will find these people in your company, but you may also find it necessary to hire someone outside.)

The second and third questions, when will the implementation be done and how, may be answered by your general (installation) plan, which is discussed next.
GENERAL PLANNING

The General Installation Plan is made up of two items: the Activity List (Figure 05.1) and the Activity Time Estimates (Figure 05.2)

Your Activity List contains the major areas of concentration. It answers the questions "who" and "what". Your Activity Time Estimates answers the question "when". However, you still do not have enough detail.

Before going into more detail, go back and be sure the two lists are fully understood.

The Activity List contains the major installation activities you need to complete a successful installation. The first two areas, Installation Organization and Document Current Processes, although not end products, are most important. They are the foundation of your installation. The remaining items on the list are:

- Application Design
- Operations Planning
- Physical Planning
- Conversion and Applications Complete
- Evaluation

These will go smoothly if you ensure that the first two areas are complete.

Your Activity Time Estimates makes this point clear; notice that the early parts of your installation efforts, as mentioned previously, must all have start dates. If your foundation is firm and on schedule, the later installation activities will also be smooth and on schedule.

The later installation activities require more detail. You may find these items helpful in planning applications other than those listed.
### Application Development Plan

#### Activity Time Estimates

<table>
<thead>
<tr>
<th>Activity</th>
<th>Duration in Weeks</th>
<th>&quot;Must&quot; Start (S) or Finish (F) Date</th>
<th>Original Schedule Dates</th>
<th>Revised Dates #1</th>
<th>Revised Dates #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Start</td>
<td>Finish</td>
<td>Start</td>
</tr>
</tbody>
</table>

Figure 05.2.
APPLICATION AND CONVERSION PLANNING

Figure 05.3 is the Activity List for your Application Development Plan. This corresponds to the Activity List for your General Installation Plan. Similarly, Figure 05.4 is the Activity Time Estimates for your Application Development Plan.

The Application Development Plan is, in general, composed of three items:

1. Analysis
   a. Review of present system
   b. Designing reports and card layouts
   c. Flowcharting
2. Evaluation
   a. Establishment of controls
   b. Management review
3. Programming of the application

The most important steps in this process are, once more, the earliest: Analysis and Evaluation. If these items are complete, that is, if the individuals and groups involved agree with what you propose, the remainder of the installation effort will be relatively free from serious problems.

Figures 05.5 and 05.6 are, respectively, the Activity List and Activity Time Estimates for the Conversion Plan.

Notice that the discussion of the Application Development Plan so far has not included programming. The question, how will the programming be carried to completion, will be discussed next.
APPLICATION DEVELOPMENT PLAN

ACTIVITY LIST

For each application:
- Review present system
- Design reports and card layouts
- Flowchart
- Establish controls
- Management review
- *Program development
- *Further detail

Figure 05.3.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Duration in Weeks</th>
<th>&quot;Must&quot; Start (S) or Finish (F)</th>
<th>Original Schedule Dates</th>
<th>Revised Dates #1</th>
<th>Revised Dates #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Date</td>
<td>Start</td>
<td>Finish</td>
<td>Start</td>
</tr>
<tr>
<td>Payroll and Labor Distribution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review present system</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design reports and card layouts</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Establish controls</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Management review</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*Program development</td>
<td>7.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accounts Receivable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review present system</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design reports and card layouts</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Establish controls</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Management review</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*Program development</td>
<td>5.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accounts Payable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review present system</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design reports and card layouts</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Establish controls</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Management review</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*Program development</td>
<td>6.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sales Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review present system</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design reports and card layouts</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Establish controls</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Management review</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*Program development</td>
<td>4.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inventory Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review present system</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design reports and card layouts</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Establish controls</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Management review</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*Program development</td>
<td>7.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Further detail (Figure 05.8)
CONVERSION PLAN

ACTIVITY LIST

For each application (where applicable):
- Develop conversion procedures
- Train conversion personnel
- Convert files
- Parallel or pilot run
- Train other departments

Figure 05.5

<table>
<thead>
<tr>
<th>Activity</th>
<th>Duration in Weeks</th>
<th>&quot;Must&quot; Start (S) or Finish (F)</th>
<th>Original Schedule Dates</th>
<th>Revised Dates #1</th>
<th>Revised Dates #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Develop data preparation and card punching procedures</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop conversion control plans and procedures</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train conversion personnel</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convert payroll and labor distribution files</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convert accounts receivable files</td>
<td>4.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convert accounts payable files</td>
<td>4.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convert inventory files</td>
<td>6.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train other departments -- all applications</td>
<td>4.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel runs -- payroll and labor distribution</td>
<td>4.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel runs -- accounts receivable</td>
<td>4.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel runs -- accounts payable</td>
<td>4.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel runs -- inventory control</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL CONVERSION</td>
<td>10.0</td>
<td>(F) - 4/8/68</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 05.6.
PROGRAMMING PLANNING

The Activity List and Activity Time Estimates for the Program Development Plan (Figures 05.7 and 05.8 respectively) complete the planning.

This is the detailed level of planning on which your installation depends. For this reason you must have control over the progress of these activities. The Progress Charts for Program Development (Figure 05.9) will provide the necessary control. Used in conjunction with the Activity Time Estimates for the Program Development Plan, these charts show you, at all times, the progress of your installation effort. You can determine whether it is ahead of schedule, on schedule, or behind schedule and requiring action.

---

**PROGRAM DEVELOPMENT PLAN**

**ACTIVITY LIST**

For each application:
- Define program
- Flowchart
- Code
- Desk-check and list
- Prepare test data
- Test
- Production test
- Complete program documentation

Figure 05.7.
<table>
<thead>
<tr>
<th>Activity</th>
<th>Duration in Weeks</th>
<th>&quot;Must&quot; Dates</th>
<th>Original Schedule Dates</th>
<th>Revised Dates #1</th>
<th>Revised Dates #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define PAY 01 (Payroll)</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart PAY 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code PAY 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list PAY 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data PAY 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test PAY 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test PAY 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation PAY 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define PAY 02 (Payroll)</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart PAY 02</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code PAY 02</td>
<td>.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list PAY 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data PAY 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test PAY 02</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test PAY 02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation PAY 02</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define PAY 03 (Payroll)</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart PAY 03</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code PAY 03</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list PAY 03</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data PAY 03</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test PAY 03</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test PAY 03</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation PAY 03</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define PAY 04 (Payroll)</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart PAY 04</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code PAY 04</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list PAY 04</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data PAY 04</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test PAY 04</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test PAY 04</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation PAY 04</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define PAY 05 (Payroll)</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart PAY 05</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code PAY 05</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list PAY 05</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data PAY 05</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test PAY 05</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test PAY 05</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation PAY 05</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define PAY 06 (Payroll)</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart PAY 06</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code PAY 06</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list PAY 06</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data PAY 06</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test PAY 06</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test PAY 06</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation PAY 06</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define PAY 07 (Payroll)</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart PAY 07</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code PAY 07</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list PAY 07</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data PAY 07</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test PAY 07</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test PAY 07</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation PAY 07</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Only one start and finish date should be supplied for each program being developed.

Figure 05.8 (Sheet 1 of 5).
<table>
<thead>
<tr>
<th>Activity</th>
<th>Duration in Weeks</th>
<th>&quot;Must&quot; Start (S) or Finish (F) Dates*</th>
<th>Original Schedule Start</th>
<th>Original Schedule Finish</th>
<th>Revised Dates # 1* Start</th>
<th>Revised Dates # 1* Finish</th>
<th>Revised Dates # 2* Start</th>
<th>Revised Dates # 2* Finish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define PAY 08 (Payroll)</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart PAY 08</td>
<td>.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code PAY 08</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list PAY 08</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data PAY 08</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test PAY 08</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product test PAY 08</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation PAY 08</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define PLD 01 (Labor Dist.)</td>
<td>.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart PLD 01</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code PLD 01</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list PLD 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data PLD 01</td>
<td>.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test PLD 01</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test PLD 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation PLD 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define PLD 02 (Labor Dist.)</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart PLD 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code PLD 02</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list PLD 02</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data PLD 02</td>
<td>.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test PLD 02</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test PLD 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation PLD 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define AR 01 (Accts Rec)</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AR 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AR 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AR 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AR 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AR 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AR 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AR 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define AR 02 (Accts Rec)</td>
<td>.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AR 02</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AR 02</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AR 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AR 02</td>
<td>.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AR 02</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AR 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AR 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define AR 03 (Accts Rec)</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AR 03</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AR 03</td>
<td>.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AR 03</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AR 03</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AR 03</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AR 03</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AR 03</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define AR 04 (Accts Rec)</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AR 04</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AR 04</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AR 04</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AR 04</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AR 04</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AR 04</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AR 04</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Only one start and finish date should be supplied for each program being developed.
<table>
<thead>
<tr>
<th>Activity</th>
<th>Duration in Weeks</th>
<th>&quot;Must&quot; Start</th>
<th>Start</th>
<th>Finish</th>
<th>Revised Dates = 1</th>
<th>Revised Dates = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define AR 05 (Accts Rec)</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AR 05</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AR 05</td>
<td>.7</td>
<td>.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AR 05</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AR 05</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AR 05</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AR 05</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AR 05</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define AR 06 (Accts Rec)</td>
<td>.5</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AR 06</td>
<td>.5</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AR 06</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AR 06</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AR 06</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AR 06</td>
<td>.4</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AR 06</td>
<td>.4</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AR 06</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define AP 01 (Accts Pay.)</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AP 01</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AP 01</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AP 01</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AP 01</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AP 01</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AP 01</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AP 01</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define AP 02 (Accts Pay.)</td>
<td>.5</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AP 02</td>
<td>.3</td>
<td>.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AP 02</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AP 02</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AP 02</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AP 02</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AP 02</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AP 02</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define AP 03 (Accts Pay.)</td>
<td>.4</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AP 03</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AP 03</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AP 03</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AP 03</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AP 03</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AP 03</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AP 03</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define AP 04 (Accts Pay.)</td>
<td>.5</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AP 04</td>
<td>.4</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AP 04</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AP 04</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AP 04</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AP 04</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AP 04</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AP 04</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define AP 05 (Accts Pay.)</td>
<td>.4</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AP 05</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AP 05</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AP 05</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AP 05</td>
<td>.1</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AP 05</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AP 05</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AP 05</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Only one start and finish date should be supplied for each program being developed.
# PROGRAM DEVELOPMENT PLAN

## ACTIVITY TIME ESTIMATES

<table>
<thead>
<tr>
<th>Activity</th>
<th>Duration in Weeks</th>
<th>&quot;Must&quot; Start (S) or Finish (F) Dates</th>
<th>Original Schedule Dates</th>
<th>Revised Dates #1</th>
<th>Revised Dates #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define AP 06 (Accts Pay.)</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AP 06</td>
<td>.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AP 06</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AP 06</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AP 06</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AP 06</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AP 06</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AP 06</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define AP 07 (Accts Pay.)</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart AP 07</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code AP 07</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list AP 07</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data AP 07</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test AP 07</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test AP 07</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation AP 07</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define INV 01 (Inventory)</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart INV 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code INV 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list INV 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data INV 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test INV 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test INV 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation INV 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define INV 02 (Inventory)</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart INV 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code INV 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list INV 02</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data INV 02</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test INV 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test INV 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation INV 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define INV 03 (Inventory)</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart INV 03</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code INV 03</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list INV 03</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data INV 03</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test INV 03</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test INV 03</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation INV 03</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define INV 04 (Inventory)</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart INV 04</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code INV 04</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list INV 04</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data INV 04</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test INV 04</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test INV 04</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation INV 04</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define INV 05 (Inventory)</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart INV 05</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code INV 05</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list INV 05</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data INV 05</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test INV 05</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test INV 05</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation INV 05</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Only one start and finish date should be supplied for each program being developed.*

Figure 05.8 (Sheet 4 of 5).
### PROGRAM DEVELOPMENT PLAN

#### ACTIVITY TIME ESTIMATES

<table>
<thead>
<tr>
<th>Activity</th>
<th>Duration in Weeks</th>
<th>&quot;Must&quot; Start (S) or Finish (F) Dates*</th>
<th>Original Schedule Dates</th>
<th>Revised Dates ≠ 1*</th>
<th>Revised Dates ≠ 2*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define INV 06 (Inventory)</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart INV 06</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code INV 06</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list INV 06</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data INV 06</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test INV 06</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test INV 06</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation INV 06</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define SA 01 (Sales Anal.)</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart SA 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code SA 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list SA 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data SA 01</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test SA 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test SA 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation SA 01</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define SA 02 (Sales Anal.)</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart SA 02</td>
<td>.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code SA 02</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desk-check, list SA 02</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test data SA 02</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test SA 02</td>
<td>.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production test SA 02</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete documentation SA 02</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total, application development</td>
<td>16.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Only one start and finish date should be supplied for each program being developed.

**Figure 05.8 (Sheet 5 of 5).**

### PERCENTAGE COMPLETED

#### Payroll

<table>
<thead>
<tr>
<th>Activity</th>
<th>Start 11/20</th>
<th>Start 12/4</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Payroll</td>
<td>PAY 01</td>
<td>PAY 02</td>
</tr>
<tr>
<td>Define program</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Document logic</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Code</td>
<td>100</td>
<td>70</td>
</tr>
<tr>
<td>Desk-check</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Prepare test data</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Test</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Production test</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Complete documentation</td>
<td>90</td>
<td>20</td>
</tr>
<tr>
<td>All activities above</td>
<td>85</td>
<td>49</td>
</tr>
</tbody>
</table>

**Figure 05.9. Program development -- progress chart.**
Section 10: DOCUMENTING CURRENT APPLICATIONS

CONTENTS

<table>
<thead>
<tr>
<th>Introduction</th>
<th>10.01.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation of Manual Systems</td>
<td>10.10.00</td>
</tr>
<tr>
<td>Documentation of Punched Card</td>
<td>10.20.00</td>
</tr>
<tr>
<td>Systems</td>
<td>10.30.00</td>
</tr>
<tr>
<td>Accounting Controls</td>
<td>10.40.00</td>
</tr>
<tr>
<td>Survey Questionnaires</td>
<td>10.40.10</td>
</tr>
<tr>
<td>Billing</td>
<td>10.40.20</td>
</tr>
<tr>
<td>Accounts Receivable</td>
<td>10.40.30</td>
</tr>
<tr>
<td>Sales Analysis</td>
<td>10.40.40</td>
</tr>
<tr>
<td>Accounts Payable</td>
<td>10.40.50</td>
</tr>
<tr>
<td>Payroll</td>
<td>10.40.60</td>
</tr>
</tbody>
</table>

Manual System Documentation

Example - Payroll                  | 10.50.00 |
Introduction                          | 10.50.01 |
Job Description                       | 10.50.10 |
Survey Form                           | 10.50.20 |
Sample Documents                      | 10.50.30 |
Systems Flowchart of Weekly Procedure | 10.50.40 |
INTRODUCTION

Since the cornerstone of your installation effort, planning, is now complete, the time to begin documenting is at hand.

If you were going to remodel a building, it would be very important to have the plans of the structure on which you would be working. You could, of course, do the job without the plans, but much time would be wasted in trial and error as you proceeded.

The same situation exists when you are converting an application to the 1130. Proper documentation of the present system will guide you rapidly and efficiently into the new solution. Rather than spending your time "rediscovering" the old procedures, you can spend it in improving them.

Depending on whether you are converting from a manual system or a punched card system, one of the following two subsections will help you plan this phase of your preinstallation effort:

- Documentation of Manual Systems (10.10.00)
- Documentation of Punched Card Systems (10.20.00)

These introductory subsections are followed by a discussion of the ways in which your current accounting controls can be documented (10.30.00). Questionnaires used for documenting manual systems are then illustrated (10.40.00).

A payroll example, which is used also in later sections, is introduced in 10.50.00. This consists of:

- Job description
- Survey forms
- Sample documents
- Systems flowchart
DOCUMENTATION OF MANUAL SYSTEMS

Follow these steps if you currently do not use punched card equipment, or if you are planning to put additional applications on the computer that are not now mechanized:

1. Ask questions about details of the job as it is being done now.
2. Record the procedure by means of a flowchart.
3. Gather samples of all the documents being used.

Survey Notes. A set of questionnaires (10.40.00) is included that assists in surveying the most common data processing procedures: billing, accounts receivable, sales analysis, inventory, accounts payable, and payroll. No questionnaire can cover all the details of, for instance, all billing procedures, but a start can be made that will lead you to discover and analyze the unique elements that have to be accounted for in your own system. Before starting your survey with a questionnaire, review the questions and determine which ones you already know the answers to, those you want to check out, and those you know are not applicable to your company. Then add questions of your own.

Where none of these survey questionnaires are applicable, record on plain paper the important elements of the system, answering the questions "who", "what", "when", "why", and "how". Notice the amount of detail called for by the questionnaires, and get down to that level in your own surveys.

You will often find that the people most familiar with the details of the job do not see the forest for the trees, or -- to use a more precise metaphor -- they think they have been looking only at elms when some of the trees have been maples. Wherever possible, count the files yourself (rough counts are usually adequate), look at the completed (not the blank) documents, and talk to the man who actually does the work, rather than taking someone else's word for what he does.

Flowcharts. As an understanding of the procedure is developed, you should draw flowcharts in which input/output documents are represented by one kind of block, processing or handling steps by another, and the flow of work by arrows, as shown in Figure 10.1.

Other symbols can be used for certain variations of the basic symbols; these are discussed in greater detail in the IBM manual Flowcharting Techniques (C20-8152) and illustrated in the manual examples in this section.

Sample Documents. Samples of each document used in the procedure should be gathered. Where possible, filled-in documents should be picked up, as well as blank documents on which the people closest to the work have made notes explaining how the documents are completed.

In other words, you should have at least two samples of each document in the system:

1. A blank document. This should have the following information written on it:
   a. The volume of these documents produced each day, week, or month -- both maximum and average.
   b. Who produces them.
   c. Where they come from, and where they go, copy by copy.
   d. For each different kind of information, or "field", all possible varieties of information that can be entered. State how long the field must or can be, and whether
each individual "position" or character in the field is strictly numeric, is sometimes alphabetic, may be blank, may contain special characters $.,'*()=+-&/, or has any other restrictions on it.

e. For each field, whether the information in it has limits. For instance, a weekly salary field could go up to $999.99 and still consist of only five digits, but you may want to pull out all of those that go above $500.00 for special handling.

f. For each field, its origin. If it has been calculated, show the formula. If it came from another document, state which one, and whether it has been altered in the process. Beware of fields that have the same name but are slightly different, such as date of receipt, date of entry, date of transcription, date of processing.

2. At least one filled-in document. The filling-in should be done by the man who normally performs the job, and he or you should annotate the reasons for and restrictions on each step of his work. Make sure that all possible ways of filling in the document have been illustrated.

Summary. When the documentation of manual systems has been completed, you should have at hand:

1. Flowcharts
2. Sample documents
3. Survey notes, including:
   a. Complete lists of codes
   b. Current standards
   c. Procedure descriptions, where the flowchart is not self-explanatory
   d. Reasons for current methods
   e. Accounting control procedures
   f. Any other facts they may influence or cause restrictions on the way an application may be designed.

All these survey notes should be cross-referenced to the flowcharts and sample documents.
DOCUMENTATION OF PUNCHED CARD SYSTEMS

Follow these steps in documenting your present punched card applications:

1. Make a list of all your control panels.
2. Arrange the list by job step within application. For instance, a payroll application, like the one in 10.50.00, might consist of panels to perform the balancing of current earnings cards to time cards, matching current deductions cards to earnings cards, preparing the deduction register, and all the remaining job steps.
3. Obtain copies of all the reports that have been run using these panels.
4. Collect your current spacing charts and card layouts and make a checklist of them. Use your list of control panels to make sure that you have gathered spacing charts and card layouts for all the operations. If not, put them on your checklist, and either find them or get them made up.
5. Check your spacing charts against the currently run copies of your reports, and bring your spacing charts up to date. Mark them on your checklist as they are updated.
6. Check your card layouts against your procedures as you run them. This will allow you to update both the card layouts and the written procedures to conform with your current actual practice. Mark the card layouts on your checklist as they are updated.
7. Obtain a current schedule of jobs. Use your list of control panels to verify the schedule.

Having finished these steps, you should have current and accurate copies of spacing charts and card layouts. If you do not, your 1130 application design and program development will suffer, and you will be forced to retrace your steps to get updated facts. The surveys (in 10.40.00) will either verify the accuracy of your documentation or indicate discrepancies that need to be checked further.

Next, since you have all the information at hand, you can develop the following items:

1. Updated flowcharts of your applications
2. Job descriptions
3. Calculation descriptions and formulas

These items, if prepared thoroughly (and this is a very important "if"), can serve as the basis for your entire 1130 application design effort.

Summary. The important thing in documenting any procedure is that all the information be made available to the programmer in concise, easily understood form.

You will find that these documenting methods will be very useful in analyzing all the procedures in your business. By pinpointing bottlenecks, areas of duplication, etc., they can provide a means of improving those procedures that you do not plan to convert immediately to the new system.

Once a program has been completed for an application, the documentation will become a permanent record of the procedure. It can be used, for example, as:

1. A source of information for implementing future changes.
2. An education device for familiarizing new operators and management personnel with the procedures.
3. A source of information for your auditors, who must be familiar with your procedures.

Start documenting your present applications now. Once the application is documented, programmed, and operating on your new system, keep the documentation up to date. It will contribute toward an efficient and productive data processing installation.
ACCOUNTING CONTROLS

Understanding your present controls will help you design practical and effective controls for your new system.

Control procedures can be documented in two places:

1. On your flowcharts, where, for instance, control tapes are balanced to accounting machine totals.
2. With the survey questionnaires or informal narratives.

For a discussion of various kinds of accounting controls that may appear in your system, refer to section 20.10.00.
SURVEY QUESTIONNAIRES

Survey Questionnaire - Billing

PROCEDURES
1. Bill before shipment or after?
2. Reasons
3. Is completion billing used?
4. Optimum time from order to shipment
5. Are shipments from stock? What percent?
   (a) Buy outside %
   (b) Manufacture? Drop Ship?
6. Do you send confirmation of order to customer? When?
7. Sold-to and ship-to information required on invoices? % of invoices?

TERMS
1. Standard by customer, variable by customer, or other
2. Do salesmen have protected customers?
3. Pricing flexible - changed to meet competition in field?
4. How many must be acknowledged?
5. Cash sales - volume and how handled?

ITEM QUANTITIES
1. Whole numbers, fractions, or decimals?
2. Will you print quantity ordered, quantity shipped, back ordered?
3. Largest quantity sold (include decimals)
4. Unit of issue

PRICES
1. Standard, volume determines, customer class, variable? How many prices?
2. Percent of billing lines with variable pricing daily
Billing Questionnaire (cont'd)

3. Variable pricing authorized by?

4. Per CM, dz, gross, bd ft, other?

5. Largest unit price

6. Fractional prices

DISCOUNTS

1. Line item only? Variable or standard?

2. What governs discounts to customers?
   (a) Customer
   (b) Type of merchandise
   (c) Quantity of merchandise
   (d) Salesman's quoted price
   (e) Total of invoice
   (f) Combination of above
   (g) Other

3. Group discounts

4. Discounts on total invoice?
   (a) Standard by customer
   (b) Variable

5. Should discount amount print on invoice?

6. Chain discounts?
   (a) Line items
   (b) Groups
   (c) Invoice totals

7. Chain discount examples

8. Terms or cash discount. Should it be calculated?
Billing Questionnaire (cont'd)

COSTING
1. Standard, percent, other?
2. Any lot or job costs?

TAXES
1. How many states?
2. What % of items taxable?
3. Are selected items on an invoice taxable?
4. Other taxes - excise, etc.
5. Whole percents, fractional?

FREIGHT
1. Based upon weight? Volume? Explain
2. Examples of computation
3. Prepaid percent - collect percent
4. Is freight cost known at billing time?
5. At prebilling time?
6. Allowances - examples. How computed?
7. Flat rates?
8. Minimums?
9. Do items have standard weights?

COMMISSIONS
1. Paid on:
   (a) Gross profit
   (b) Gross invoice
   (c) Variable each line
Billing Questionnaire (cont'd)

(d) Total customer purchase

(e) Other

2. Percentage fixed by:

(a) Product?

(b) Salesman?

(c) Customer?

(d) Volume?

3. If volume, what are the breaks in computing rate?

FORMS INFORMATION

1. Use of copies

   1

   2

   3

   4

   5

   6

   7

   8

   9

  10

2. If you prebill, could invoice serve as picking document? As bill of lading?

3. Average number of body lines

4. Minimum depth of form

5. Preprint invoice number? Why?

6. Are back orders noted on invoice?

7. What is the length of item descriptions?

   (a) Number and type of special characters included in descriptions?

   (b) Can description be conveniently abbreviated?

8. Discount on line item?

9. Largest quantity shipped? Largest unit price? Largest extension?
Billing Questionnaire (cont'd)

10. Cash discount printed on invoice? Terms?

11. Length of ship-to/sold-to lines

12. Cost extended - line items?

13. Do credit memos and invoices use same format?

14. Are contractual notes typed on invoice or credit memo?
   (a) If yes, what is longest note?
   (b) What is the incidence of use (percent) of total invoices per day?

15. Multi-page invoice? Frequency

16. What class of products is most active? At what time of the year?

17. Are the products of a seasonal nature? When? What is increase in orders?

18. What items make up largest percentage of total sales volume?

19. How much item information is needed on the order? On the invoice? Can it be typed later?

20. How are items coded?
   (a) What is the length of part number or code?
   (b) Numeric or alphanumerical?

21. What procedure is being followed as to partial shipments?
   (a) How prevalent are they?
   (b) Are shipments made daily to all areas? If not, what is the policy regarding shipments?
   (c) Is warehouse sequence of items on the order important?

22. Describe miscellaneous data required.
Billing Questionnaire (cont'd)

ANALYSIS

1. Time from receipt of order to billing of customer

2. Number and jobs of people performing order writing and billing

3. Type of machines and equipment presently being used

CONTROL AND EDITING INFORMATION

1. What is the editing procedure for invoicing? Who is responsible for final approval of invoice?

2. What controls are now established for accuracy?

3. Do you have subsidiary branch locations?
   (a) If so, what accounting functions are they performing?
   (b) How many invoices is each branch preparing?
   (c) Would it be more advantageous to centralize accounting operations, especially billing?
Survey Questionnaire – Accounts Receivable

PROCEDURES: CASH

1. List all cash credit posting media

2. What discounts are offered? How are they handled?

3. Cash receipts and deposit slip prepared:
   (a) Separately
   (b) Simultaneously

4. How often do payments include copy of invoice or statement or identification?

5. What percentage of payments are nonstandard?

6. What is policy on overpayments?

7. Can cash be applied to oldest balance or must it be selective?

8. What accounts are involved?

9. Can distribution be made at cash posting time?

10. How many ledger controls are carried?
    (a) How are control groups determined?
    (b) Illustrate divisions

11. How often is a trial balance taken?
    (a) Can trial balance be alternated by control?
    (b) Could trial balance, aging, and customer purchasing analysis be prepared simultaneously?

12. When are statements mailed?

13. Attach samples of accounting (A/C) journal used, revised to include additional information you require.

14. Volume and reasons for credit memos
Accounts Receivable Questionnaire (cont'd)

FORMS CONSIDERATIONS - STATEMENTS

1. How many accounts in ledgers?
   (a) Total active
   (b) Total inactive
   (c) Does total fluctuate or remain static?
   (d) How are they coded?

2. Open item or balance forward?

3. What percent of customers pay by:
   (a) Statement?
   (b) Invoice?
   (c) Time pay?

4. How many statements mailed?
   (a) Total
   (b) Weekly
   (c) Monthly
   (d) Are they mailed to all accounts?

5. If time pay is allowed, explain circumstances.

6. Do statements show:
   (a) All transactions for the month?
   (b) Open items only?
   (c) Aged balances only?
   (d) Aged transactions?

7. Any objection to aged balances only, with no reference?

8. What description shows on statement?
Accounts Receivable Questionnaire (cont’d)

9. Daily inquiries into customer records?

10. Extent of bad debts

11. Attach a sample statement, complete with various postings.

LEDGER RECORDS

1. What description is shown on ledgers?

2. Credit limit on each card?

3. Purchases to date? Is this desirable?

4. Is aging by invoice? Oldest dollar amount?

5. Attach a sample card, complete with typical postings.

CREDIT REFERENCE

1. Does credit department refer to ledgers? How often?

2. Is a credit record other than ledger kept? If so, attach a sample.

3. When does an account become delinquent?

4. How are delinquents followed?

5. Do you suspend credit buying of delinquent accounts? If so, how is it restored?

6. Are accounts aged?
   (a) What breakdowns?
   (b) When?
   (c) How often?

ANALYSIS

1. Number of people involved

2. Type of equipment involved
Survey Questionnaire - Sales Analysis

1. Information required by:
   (a) Customer
   (b) Item
   (c) Area
   (d) Salesman
   (e) Class of trade

2. What reports should management be receiving that they are not now getting?

3. Report information
   (a) What information is required on each report?
      (1) What records or registers are used to substantiate reports?
      (2) What can be added to present reports to make them more meaningful?
   (b) Who receives each report?
   (c) By what priorities are reports prepared?
   (d) Are cost analysis reports generated?
      (1) How often?
      (2) To whom?
      (3) What information?
      (4) By what classification?
   (e) Are gross reports prepared?
      (1) By what classification?
   (f) Are comparative sales analysis reports generated?
      (1) What period are the results based on?
   (g) Are salesman commission statements prepared?
      (1) How many salesmen?
Sales Analysis Questionnaire (cont'd)

4. Control information

   (a) What are controls and editing procedures for above reports?

5. What is present cost to derive these reports?
Survey Questionnaire - Inventory

1. What percentage of inventory items account for:
   (a) High activity? %
   (b) Medium activity? %
   (c) Low activity? %

2. Does the present coding structure have any real significance, such as block code, significant digit, etc.? 
   (a) Give example
   (b) Are bin locations assigned in sequence by part number?

3. How many transactions are there of each type?
   (a) Receipts and returns
   (b) Issues
   (c) Miscellaneous

4. Are standard or economic order quantities used? If so, how are they determined?
   (a) Do you order by vendor group or as required?

5. Does the inventory record reflect planned requirements, such as:
   (a) On-hand balance
   (b) On-order balance
   (c) Reserved balance
   (d) Available balance
   (e) Minimum balance
   (f) Usage data, etc.
   (g) Maximum balance

6. What inventory costing method is used?
   (a) Average
   (b) Last in, first out (LIFO)
Inventory Questionnaire (cont'd)

(c) First in, first out (FIFO)

(d) Standard

7. What is the frequency of inventory cost changes? What is the frequency of inventory sales price changes?
   (a) How often are price changes of finished goods made?
   (b) Are they made by product line or by item?

8. If partial shipments are made, what is the procedure for handling them?

9. Is there a back-order problem? If so, how is it controlled?
   (a) What percentage of orders have items back-ordered, substituted or canceled?
   (b) How much $ volume do you lose?

10. How and when is a physical inventory taken? By whom?

11. What controls are set up and maintained on the inventory system?

12. What is the cost of inventory maintenance?

13. What are the present costs of keeping inventory records?

14. What are the types of inventory records and reports?
   (a) Do they result in a stock status summary report?
   (b) How often are inventory reports prepared?
   (c) Who receives them?

15. What is the origin and layout of source documents and what controls are used?

16. How often are inquiries made into inventory records? What are their nature? Who makes them?

17. How are present inventory recordkeeping functions correlated with purchasing, billing, sales, manufacturing, etc.?
Inventory Questionnaire (cont'd)

18. What comparative information do you need?
   (a) Month-to-date
   (b) Year-to-date
   (c) Same period last year
   (d) Percent of comparisons

19. Where must current inventory records be physically located?
Survey Questionnaire - Accounts Payable

REPORT INFORMATION

1. Is a cash requirement register being prepared?
   (a) What is the average daily cash requirement to meet payables?
   (b) How often is this register prepared?

2. Are amounts being distributed and charged to job orders and expense accounts?
   (a) What is the procedure for each of the above?
      (1) Number of open job orders
      (2) Number of expense accounts
   (b) Are departments budgeted?
      (1) How often are budgets depleted and how often are analysis reports submitted?

CONTROLS AND EDITING PROCEDURES

1. How are payable accounts reconciled?

2. Who is responsible for editing before releasing checks, and what is the procedure?

3. How often are payable accounts reviewed?

4. What controls are in effect?

PURCHASES

1. Number of vendors active and inactive. What are criteria for active?

2. Are orders placed verbally, by requisition, by purchase order, or other?

3. Is blanket order placed for staggered shipments?

4. How are incoming goods accounted for?

5. How are partial shipments handled?

6. What method is used to notify Accounts Payable regarding overs, shorts, or damaged goods?
Accounts Payable Questionnaire (cont'd)

7. Are purchase orders (P. O.'s) coded by Accounting when written?
   (a) If not, when and how are codes assigned?

INCOMING INVOICES

1. Is an invoice register maintained? If not, how are invoices controlled?
2. Pay by statement?
   (a) Is early-pay discount given?
3. When is liability recognized?
   (a) Receipt of goods
   (b) Receipt of invoice
4. Are invoices matched to P. O.'s?
5. Are invoices received from same vendor with different discount dates? How are they handled?
6. Are any invoices paid before arrival of goods?
7. Can one invoice be charged to two or more accounts?

PROCEDURE

1. Is a voucher system presently in use? Ledger system? Other?
2. How are invoices or vouchers filed to ensure that discounts will be taken?
3. Are incoming invoices numbered consecutively?
   (a) Upon receipt?
   (b) Other?

CHECK WRITING

1. How many banks are checks drawn against?
2. If more than one, can the bank be determined before the voucher is opened?
3. Are checks prenumbered?
Accounts Payable Questionnaire (cont'd)

5. How often are checks written?

6. What is present form of checks, voucher, and remittance advice? Attach sample.

7. Are discounts computed at check-writing time? If not, when?

8. Is a check register required?

9. Are certain checks written daily? If so, estimate number.

DISTRIBUTION

1. Which accounts receive greatest number of distributions?

2. How many income and expense accounts are kept? How many divisions are used?

3. How many controlling accounts? Identify each.

4. What department or person is responsible for A/C distribution of invoice?

5. Is apron or rubber stamp used?

6. What percent of invoices contain items chargeable to different income and expense accounts?

7. Is distribution made directly from invoice? At checkwriting time?

8. How much detail in distribution record?

9. How many items other than invoices (e.g., journal vouchers) are distributed each month?

10. What is cutoff date?

11. When is trial balance secured?

12. How is trial balance secured?

MISCELLANEOUS

1. Is obligation record required?

2. Is purchase journal available? How prepared?

3. Is vendor control card required?

4. Total purchases-to-date by vendor required?

5. Do you, or will you, use group processing method?

6. Do you, or will you, use balance-forward method?

7. Are expenditures compared against budget?
Survey Questionnaire - Payroll

1. How is time figured?
   (a) Tenths of hours
   (b) Hundredths of hours
   (c) Hours and minutes
   (d) Other (nearest half or quarter hour)
   (e) Incentive or price rates

2. What is overtime?
   (a) Over 40 hours
   (b) Over 8 hours
   (c) Other

3. How prevalent are rate changes? Temporary or permanent?
   (a) How many can a man have?
   (b) When?
   (c) Does job carry a rate?

4. How many shifts are there?
   (a) What kind of bonus is there?
   (b) How is it calculated?

5. What is employee turnover?

6. What YTD information will appear on check stub?

7. How many timekeepers?

8. Are timeclocks used? Is time recorded in tenths or hundredths of hours?

9. Is there labor distribution?
   (a) By job? Department? Operation? Machine?
   (b) Is average labor cost used?
Payroll Questionnaire (cont'd)

(c) Actual labor cost?

(d) How is overtime handled?

PREPARATION DATA

1. What are pay periods?

2. When does pay period close?

3. What is paying date? Preparation time?

4. How are employees paid?
   (a) Check, cash?
   (b) Is envelope used?

5. How many copies of journals?

6. Any objection to the use of spot carbon on check?

7. Should check amount be protected?

8. Is check signer used?

9. Do you write payroll checks on more than one bank?

10. How and when are vacation checks written?

11. How are advances handled?

12. How are terminations handled?

13. How is sick pay handled?

14. How is holiday pay handled?

INCENTIVES, SHIFTS, ETC.

1. How many shifts?

2. What is incentive formula?

3. Are rates for various jobs known by employees?

4. How often is it necessary to pay "make-up" pay?

5. List indirect labor categories
**Payroll Questionnaire (cont'd)**

6. Are efficiency standards established?
   
   (a) By machine?
   
   (b) By employee?

**DEDUCTIONS**

1. Voluntary

2. Involuntary

3. Average deduction amount
   
   (a) Voluntary

   (b) Involuntary

4. Percentage of activity
   
   (a) Voluntary

   (b) Involuntary
Payroll Questionnaire (cont’d)

5. Largest month total ($)
   (a) Voluntary
   
   (b) Involuntary

6. List the posting media for each of the above

7. What reports must be furnished?

8. How are salesmen paid?
   (a) Salary or standard commission
   
   (b) Explain other
Payroll Questionnaire (cont'd)

9. Reports (payroll and labor distribution)
   (a) Form (sequence of information)
   (b) Content (size of fields, number of classifications)
   (c) Frequency (Presently? With IBM approach to application?)
   (d) Distribution

10. Schedule requirements
   (a) Length of pay period
   (b) When are source documents available for processing?
   (c) When does pay period close?
   (d) How soon after pay period closes must checks be available?
   (e) How long does it take for changes to clear through the personnel department?

11. Reporting
   (a) Who reports payroll source data? Employees? Timekeeper? Foreman?
   (b) What degree of control does the accounting department have over the people who report data?

12. Management requirements
   (a) Who gets the reports?
   (b) What would they like that their present system doesn't give them?

13. Miscellaneous
   (a) In what states do you pay payroll?
   (b) What special deduction considerations are there?
   (c) Is state or city income tax deducted?
MANUAL SYSTEM DOCUMENTATION EXAMPLE -- PAYROLL

Introduction

This example of a typical manual application consists of the following items:

   Job Description -- Payroll
   Survey Form – filled in for payroll

   Samples of all documents being used
   Flowchart -- all of payroll procedure
   Notice that the illustrations are shown in the order in which they are ordinarily developed. After the job description is written, the survey is completed, and all sample documents are gathered. Then the procedure that produces the reports, using the information from the survey form, is drawn in flowchart form.
Job Description

A job description is not always necessary, but is useful when new people are introduced to an application, or when presentations are made for management or visitors. Both of these situations occur frequently during the conversion process.

The following is a typical job description. Note that it is short, describes objectives, and provides a summary of the procedure.

---

Payroll -- Job Description

The objectives of the payroll procedure are:

1. To record earnings, deductions, and taxes for historical purposes.

2. To provide state and federal governments, unions, and other agencies with a record of moneys collected for them.

3. To furnish employees with a personal record of earnings, deductions, and taxes.

4. To write and reconcile paychecks.

5. To provide entries to labor statistics and miscellaneous reports.

To accomplish the above, current period time cards, containing hours worked, are matched to the production report, and gross earnings are calculated and posted to the payroll register. Then, deductions and net pay are calculated and posted to the payroll register, paychecks are written, and earnings records are updated. Miscellaneous reports are produced from earnings records, and quarter-to-date information is prepared for 941 and W-2 forms preparation.
Survey Form

The following is a typical completed survey form. Note that the answers are short and descriptive. The survey form is always necessary.
FACTORY PAYROLL

Survey Questionnaire - Payroll

1. How is time figured?
   (a) Tenths of hours
   (b) Hundredths of hours X
   (c) Hours and minutes
   (d) Other (nearest half or quarter hour)
   (e) Incentive or price rates

2. What is overtime?
   (a) Over 40 hours X
   (b) Over 8 hours X
   (c) Other

3. How prevalent are rate changes? Temporary or permanent?
   (a) How many can a man have? Maximum, 10 per year
   (b) When? At union contract time, promotions
   (c) Does job carry a rate? Yes

4. How many shifts are there? 1, 2, or 3, by plant
   (a) What kind of bonus is there? 2nd $0.06
   (b) How is it calculated? Added to base and treated as rate change

5. What is employee turnover? 25% average, overall plants

6. What YTD information will appear on check stub? None

7. How many timekeepers? One per plant

8. Are timeclocks used? Yes
   Is time recorded in tenths or hundredths of hours? Tenths

9. Is there labor distribution? Yes
   (a) By job? Department? Operation? Machine?
   (b) Is average labor cost used? Yes
(c) Actual labor cost? No.
(d) How is overtime handled? As separate item.

PREPARATION DATA
1. What are pay periods? Weekly.
2. When does pay period close? Sunday.
   Following Friday.
3. What is paying date? Preparation time? 12 man-days.
4. How are employees paid?
   (a) Check, cash?
   (b) Is envelope used? Varies by plant.
5. How many copies of journals? One.
6. Any objection to the use of spot carbon on check? No.
7. Should check amount be protected? Yes.
8. Is check signer used? Yes.
9. Do you write payroll checks on more than one bank? Yes.
10. How and when are vacation checks written? Just before plant closes for vacation.
11. How are advances handled? None.
12. How are terminations handled? Records are kept as prescribed by state. Insurance pay is treated as special earnings.
13. How is sick pay handled? None.
14. How is holiday pay handled? At base rate if worked day before and day after.

INCENTIVES, SHIFTS, ETC.
1. How many shifts? 1, 2, or 3, by plant.
2. What is incentive formula? 1% of piecework completed over standard.
3. Are rates for various jobs known by employees? Yes.
4. How often is it necessary to pay "make-up" pay? Never.
5. List indirect labor categories: Printing, die room, cutting die, maintenance, janitor, watchman, trained, sales and experimental, cleaning, salvage, meeting, inventory.
6. Are efficiency standards established?  
   (a) By machine? ✗  
   (b) By employee?

DEDUCTIONS

1. Voluntary
   1. Credit Union
   2. Charitable contributions
   3. Stock

2. Involuntary
   7. Union dues
   8. Federal income tax
   9. Insurance
   10. Social security tax
   11. Local tax

3. Average deduction amount
   (a) Voluntary
       1. $5.00
       2. $0.50
       3. $1.00

   (b) Involuntary
       7. $11.50
       8. $15.00
       9. $0.50
       10. $4.00

4. Percentage of activity
   (a) Voluntary
       1. 25
       2. 90
       3. 2

   (b) Involuntary
       7. 99
       8. 100
       9. 15
       10. 100
       11. 80
      12
5. Largest month total ($)  
   (a) Voluntary  
   1 $6,000  
   2 $1,000  
   3 $ 150  
   4  
   5  
   6  
   7 $ 3,000  
   8 $ 7,500  
   9 $ 180  
   10 $ 7,000  
   11 $ 1,400  
   12 
   (b) Involuntary  
   1  
   2  
   3  
   4  
   5  
   6  
   7 $ 3,000  
   8 $ 7,500  
   9 $ 180  
   10 $ 7,000  
   11 $ 1,400  
   12 

6. List the posting media for each of the above  
   1 payroll register, check, general ledger  
   2 payroll register, check, general ledger  
   3 payroll register, check, general ledger  
   4  
   5  
   6  
   7 payroll register, check, general ledger  
   8 payroll register, check, general ledger  
   9 payroll register, check, general ledger  
   10 payroll register, check, general ledger  
   11 payroll register, check, general ledger  
   12 

7. What reports must be furnished?  
   1 credit union list  
   2 union dues list  
   3 check deduction list  
   4 W-2  
   5  
   6 local tax report  
   7 check stubs  
   8 check stubs  
   9 employee earnings record  
   10  
   11 machine activity report  
   12 

8. How are salesmen paid?  
   (a) Salary or standard commission  
   $\text{Salary} + 10\% \text{ over quota}$  
   (b) Explain other
9. Reports (payroll and labor distribution)

(a) Form (sequence of information) By machine

(b) Content (size of fields, number of classifications)

- Machine (Name) 1 sq. Ft./Machine (xxxx.x) Pet Effie (xxxx.x)
- Shift (x) 10 1/2 Std. Hrs. (xxxx.x) Delay Effie (xxxx.x)
- W sq. Ft./Mach-Hrs (xxxx.x) 10 1/2 Unrated Hrs. (xxxx.x) OT Hrs (xxxx.x)
- Wt. Pieces (xxxx.x) Man-Hours (xxxx.x) Actual Hrs. (xxxx.x)

(c) Frequency (Presently? With IBM approach to application?) Weekly and monthly

(d) Distribution Accounting, plant superintendent, plant gen. mgr.,
top management

10. Schedule requirements

(a) Length of pay period Weekly

(b) When are source documents available for processing? Monday

(c) When does pay period close? Sunday

(d) How soon after pay period closes must checks be available? 5 days

(e) How long does it take for changes to clear through the personnel department? 1 day

11. Reporting

(a) Who reports payroll source data? Employees? Timekeeper? Foreman?

(b) What degree of control does the accounting department have over the people who report data? None, except for validation of time

12. Management requirements

(a) Who gets the reports? President, chairman of the board, plant managers

(b) What would they like that their present system doesn't give them? 1. Better labor distribution reporting.

2. If T & D info. carried thru each pay period

13. Miscellaneous

(a) In what states do you pay payroll? Ohio, Ind., W. Va., Tex.

(b) What special deduction considerations are there? Union initiations, notice of levy

(c) Is state or city income tax deducted? Yes
ADMINISTRATIVE PAYROLL

Survey Questionnaire - Payroll

1. How is time figured?
   (a) Tenths of hours  
   (b) Hundredths of hours
   (c) Hours and minutes
   (d) Other (nearest half or quarter hour) 
   (e) Incentive or price rates

2. What is overtime?
   (a) Over 40 hours  
   (b) Over 8 hours
   (c) Other

3. How prevalent are rate changes? 
   (a) How many can a man have? 
   (b) When? 
   (c) Does job carry a rate?

4. How many shifts are there?
   (a) What kind of bonus is there? 
   (b) How is it calculated?

5. What is employee turnover?

6. What YTD information will appear on check stub?

7. How many timekeepers?

8. Are timeclocks used? Is time recorded in tenths or hundredths of hours?

9. Is there labor distribution?
   (a) By job? 
   (b) Is average labor cost used?
(c) Actual labor cost? Yes
(d) How is overtime handled? Included

PREPARATION DATA

1. What are pay periods? Biweekly
2. When does pay period close? Every other Friday
3. What is paying date? Preparation time? 3 man-days
4. How are employees paid?
   (a) Check cash?
   (b) Is envelope used? Yes
5. How many copies of journals? One
6. Any objection to the use of spot carbon on check? No
7. Should check amount be protected? Yes
8. Is check signer used? Yes
9. Do you write payroll checks on more than one bank? Yes
10. How and when are vacation checks written? At employee option before vacation
11. How are advances handled? None
12. How are terminations handled? Records are kept as prescribed by state. Severance pay is treated as special earnings.
13. How is sick pay handled? Standard pay, nothing special
14. How is holiday pay handled? 8 hours at base rate or salary

INCENTIVES, SHIFTS, ETC.
1. How many shifts? One
2. What is incentive formula? None
3. Are rates for various jobs known by employees? No
4. How often is it necessary to pay "make-up" pay? Never
5. List indirect labor categories Janitor, watchman, meetings, trainee

(a) By machine?

(b) By employee?

DEDUCTIONS

1. Voluntary
   1 Credit Union
   2 Charitable Contributions
   3 Insurance
   4 Stock
   5
   6
   7 Federal Income Tax
   8 Social Security
   9
   10
   11
   12

2. Involuntary
   7 Federal Income Tax
   8 Social Security
   9
   10
   11
   12

3. Average deduction amount
   (a) Voluntary
      1 $10.00
      2 $10.00
      3 $10.00
      4 $2.00
      5
      6
      7 $40.00
      8 $10.00
      9 $8.00
      10
      11
      12

   (b) Involuntary
      7 $40.00
      8 $10.00
      9 $8.00
      10
      11
      12

4. Percentage of activity
   (a) Voluntary
      1 25%
      2 90%
      3 75%
      4 15%
      5
      6
      7 100%
      8 80%
      9 100%
      10
      11
      12

   (b) Involuntary
      7 100%
      8 80%
      9 100%
      10
      11
      12
5. Largest month total ($)
   (a) Voluntary
   
   1 $ 6,000
   2 $ 4,000
   3 $ 2,000
   4 $ 3,500
   5
   6
   7 $ 5,000
   8 $ 4,500
   9 $ 3,200
   10
   11
   12

   (b) Involuntary

   7 $ 5,000
   8 $ 4,500
   9 $ 3,200
   10
   11
   12

6. List the posting media for each of the above

   1 Payroll Register, Check, General Register
   2 Payroll Register, Check, General Register
   3 Payroll Register, Check, General Register
   4 Payroll Register, Check, General Register
   5
   6
   7 Payroll Register, Check, General Register
   8 Payroll Register, Check, General Register
   9 Payroll Register, Check, General Register
   10
   11
   12

7. What reports must be furnished?

   1 Credit union list
   2 Insurance deduction list
   3 Stock deduction list
   4 W-2
   5 941
   6 Local tax report
   7 Check
   8 Check stub
   9 Employee earnings record
   10 Payroll register
   11 Machine activity report
   12

8. How are salesmen paid?

   (a) Salary or standard commission
   
   Salary + 10% over quota

   (b) Explain other
9. Reports (payroll and labor distribution)
   (a) Form (sequence of information)
   (b) Content (size of fields, number of classifications)
   (c) Frequency (Presently? With IBM approach to application?)
   (d) Distribution

10. Schedule requirements
    (a) Length of pay period
    (b) When are source documents available for processing?
    (c) When does pay period close?
    (d) How soon after pay period closes must checks be available?
    (e) How long does it take for changes to clear through the personnel department?

11. Reporting
    (b) What degree of control does the accounting department have over the people who report data?

12. Management requirements
    (a) Who gets the reports?
    (b) What would they like that their present system doesn't give them?

13. Miscellaneous
    (a) In what states do you pay payroll? Ohio, W. Va., Ind., Tex.
    (b) What special deduction considerations are there? Matric of Iowa.
    (c) Is state or city income tax deducted? Yes
SAMPLE DOCUMENTS

The following is a typical collection of sample documents. Note that both blank and completed documents are present.

It is always necessary to collect all documents, both completed and blank, for your current system.
<table>
<thead>
<tr>
<th>ORDER NO.</th>
<th>CUSTOMER</th>
<th>SQ. FT.</th>
<th>NO. OUT</th>
<th>S. U.</th>
<th>RUN</th>
<th>NO. PIECES</th>
<th>MAN HRS.</th>
<th>TOTAL SET</th>
<th>MAN HRS. RUN TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TOTA尔S**

<table>
<thead>
<tr>
<th>CLOCK NO.</th>
<th>NAME</th>
<th>START</th>
<th>STOP</th>
<th>REG.</th>
<th>BONUS</th>
<th>MACH. PIECES</th>
<th>MACH HRS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|                |       |          |      |      |       |             |         |

|                |       |          |      |      |       |             |         |

|                |       |          |      |      |       |             |         |

|                |       |          |      |      |       |             |         |

|                |       |          |      |      |       |             |         |

**TOTALS**

|                |       |          |      |      |       |             |         |

|                |       |          |      |      |       |             |         |

|                |       |          |      |      |       |             |         |

|                |       |          |      |      |       |             |         |

|                |       |          |      |      |       |             |         |

|                |       |          |      |      |       |             |         |

|                |       |          |      |      |       |             |         |

Form 101
<table>
<thead>
<tr>
<th>ORDER NO.</th>
<th>CUSTOMER</th>
<th>SQ. FT.</th>
<th>NO. OUT</th>
<th>S. U.</th>
<th>RUN</th>
<th>NO. PIECES</th>
<th>MAN HRS, TOTAL SET</th>
<th>MAN HRS, RUN TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>JINES MFG</td>
<td>8000</td>
<td>8000</td>
<td>0</td>
<td>1713</td>
<td>345</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

**Totals**

<table>
<thead>
<tr>
<th>CLOCK NO.</th>
<th>NAME</th>
<th>START</th>
<th>STOP</th>
<th>REG.</th>
<th>BONUS</th>
<th>MACH.</th>
<th>DATE</th>
<th>1/11-68</th>
</tr>
</thead>
<tbody>
<tr>
<td>4011</td>
<td>R. BEDAN</td>
<td>8:00</td>
<td>4:00</td>
<td>8</td>
<td>0</td>
<td>PIECES</td>
<td>345</td>
<td>MACH HRS 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SQ.</td>
<td>8000</td>
<td>HRS. 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SET UP</td>
<td>2</td>
<td>ALLOW 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RUN</td>
<td>7</td>
<td>MAKE UP 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TOTAL</td>
<td>9</td>
<td>BONUS 0</td>
</tr>
</tbody>
</table>

Form 101
<table>
<thead>
<tr>
<th>NAME</th>
<th>MON.</th>
<th>TUES.</th>
<th>WED.</th>
<th>THURS.</th>
<th>FRI.</th>
<th>SAT.</th>
<th>TOTALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROOALONA, J.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOY, C.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRASWELT, F.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAZDEL, M.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DORLIN, J.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOLLORE, R.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIROHOSE, V.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PANUNI, D.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WALLJAMS, J.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Master Employee Time Sheet

<table>
<thead>
<tr>
<th>NAME</th>
<th>MON.</th>
<th>TUES.</th>
<th>WED.</th>
<th>THURS.</th>
<th>FRI.</th>
<th>SAT.</th>
<th>TOTALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROOALONA, J.</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>CLOY, C.</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>CRASWELT, F.</td>
<td>8½</td>
<td>8½</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>DAZDEL, M.</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>8</td>
<td></td>
<td>46</td>
</tr>
<tr>
<td>DORLIN, J.</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>FOLLORE, R.</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>9½</td>
<td></td>
<td>43½</td>
</tr>
<tr>
<td>MIROHOSE, V.</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>PANUNI, D.</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>WALLJAMS, J.</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td></td>
<td>24</td>
</tr>
</tbody>
</table>
# TIME SHEET

<table>
<thead>
<tr>
<th></th>
<th>START—STOP</th>
<th>LUNCH</th>
<th>HOURS WORKED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MON.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TUES.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WED.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>THUR.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FRI.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL FIRST WEEK</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAT.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MON.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TUES.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WED.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>THUR.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FRI.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL SECOND WEEK</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TOTAL HOURS**

CHECKED BY

APPROVED BY
## Time Sheet

**Name:** J. Doe  
**Two Weeks Ending:** 2/2/68

<table>
<thead>
<tr>
<th>Day</th>
<th>Start-Stop</th>
<th>Lunch</th>
<th>Hours Worked</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT.</td>
<td>9-12</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>MON.</td>
<td>8-6</td>
<td>12-1</td>
<td>9</td>
</tr>
<tr>
<td>TUES.</td>
<td>8-5 30</td>
<td>1230-1300</td>
<td>8 1/2</td>
</tr>
<tr>
<td>WED.</td>
<td>8-5</td>
<td>1230-1300</td>
<td>8</td>
</tr>
<tr>
<td>THUR.</td>
<td>8-5</td>
<td>12-1</td>
<td>8</td>
</tr>
<tr>
<td>FRI.</td>
<td>8-5</td>
<td>12-1</td>
<td>8</td>
</tr>
<tr>
<td><strong>Total First Week</strong></td>
<td></td>
<td></td>
<td><strong>44 1/2</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day</th>
<th>Start-Stop</th>
<th>Lunch</th>
<th>Hours Worked</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT.</td>
<td>8-5</td>
<td>12-1</td>
<td>8</td>
</tr>
<tr>
<td>MON.</td>
<td>8-5</td>
<td>12-1</td>
<td>8</td>
</tr>
<tr>
<td>TUES.</td>
<td>8-5</td>
<td>12-1</td>
<td>8</td>
</tr>
<tr>
<td>WED.</td>
<td>8-5</td>
<td>12-1</td>
<td>8</td>
</tr>
<tr>
<td>THUR.</td>
<td>8-5</td>
<td>12-1</td>
<td>8</td>
</tr>
<tr>
<td>FRI.</td>
<td>8-5</td>
<td>12-1</td>
<td>8</td>
</tr>
<tr>
<td><strong>Total Second Week</strong></td>
<td></td>
<td></td>
<td><strong>40</strong></td>
</tr>
</tbody>
</table>

**Total Hours:** 84 1/2  
**Checked By:** JAH  
**Approved By:** EJ
### PRODUCTION & LABOR REPORT

#### Week Ending [Date]

<table>
<thead>
<tr>
<th>Rate</th>
<th>Machine</th>
<th>Shift</th>
</tr>
</thead>
</table>

#### Standard Hours

<table>
<thead>
<tr>
<th>Non Rated</th>
<th>% Eff</th>
<th>Actual Hours</th>
<th>Bonus Hours</th>
<th>Delay Time</th>
<th>Non Rate Hours</th>
<th>Actual Dollars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mech.</td>
<td>Man</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Setup, Run, Total

|--------|-----------|

#### This Week

| M | T | W | T | F | S |

#### Prev. Week

| M | T | W | T | F | S |

#### To Date

| M | T | W | T | F | S |
### PRODUCTION & LABOR REPORT

**Week Ending:** 1/13/68

<table>
<thead>
<tr>
<th>Date</th>
<th>Rate</th>
<th>Machine</th>
<th>Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.50</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dr.</th>
<th>M. Pcs.</th>
<th>M. Sq. Ft.</th>
<th>Standard Hours</th>
<th>Non Rated</th>
<th>% Eff</th>
<th>Actual Hours</th>
<th>Bonus Hours</th>
<th>Delay Time</th>
<th>Non Rate Bonus</th>
<th>Actual Overtime Hours</th>
<th>Actual Dollars</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Setup</td>
<td>Run</td>
<td>Total</td>
<td></td>
<td>Mech.</td>
<td>Man</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-9</td>
<td>520 M</td>
<td>10126</td>
<td>1.0</td>
<td>6.0</td>
<td>7.0</td>
<td>1.0</td>
<td>88</td>
<td>6.0</td>
<td>14.2</td>
<td>.5</td>
<td></td>
</tr>
<tr>
<td>1-10</td>
<td>726 T</td>
<td>13502</td>
<td>.6</td>
<td>6.8</td>
<td>7.4</td>
<td>.6</td>
<td>93</td>
<td>6.8</td>
<td>13.8</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>1-11</td>
<td>431 W</td>
<td>9526</td>
<td>1.0</td>
<td>5.8</td>
<td>6.8</td>
<td>1.2</td>
<td>85</td>
<td>5.8</td>
<td>10.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-12</td>
<td>508 T</td>
<td>9972</td>
<td>.3</td>
<td>6.9</td>
<td>7.2</td>
<td>.8</td>
<td>90</td>
<td>6.9</td>
<td>15.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-13</td>
<td>631 F</td>
<td>12703</td>
<td>.5</td>
<td>7.0</td>
<td>7.5</td>
<td>.5</td>
<td>94</td>
<td>7.0</td>
<td>16.7</td>
<td>.5</td>
<td></td>
</tr>
</tbody>
</table>

|This Week| 55829 | 3.4 | 32.5 | 35.9 | 4.1 | 90 | 32.5 | 70.8 | 1.7 | 3.4 | 3.8 | 2.2 | 5687 |
|Prev. Wks.| 107501 | 7.6 | 63.3 | 70.9 | 9.1 | 89 | 67.3 | 143.0 | 5.0 | 7.5 | 6.9 | 1.7 | 3.7 | 12708 |
|To Date| 163350 | 11.0 | 95.8 | 106.8 | 13.2 | 89 | 99.8 | 2138 | 6.7 | 10.7 | 10.7 | 1.7 | 5.9 | 18395 |

### PRODUCTION & LABOR REPORT

**Week Ending:** 1/13/68

<table>
<thead>
<tr>
<th>Date</th>
<th>Rate</th>
<th>Machine</th>
<th>Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.75</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date</th>
<th>M. Pcs.</th>
<th>M. Sq. Ft.</th>
<th>Standard Hours</th>
<th>Non Rated</th>
<th>% Eff</th>
<th>Actual Hours</th>
<th>Bonus Hours</th>
<th>Delay Time</th>
<th>Non Rate Bonus</th>
<th>Actual Overtime Hours</th>
<th>Actual Dollars</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Setup</td>
<td>Run</td>
<td>Total</td>
<td></td>
<td>Mech.</td>
<td>Man</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-9</td>
<td>606 M</td>
<td>13706</td>
<td>.6</td>
<td>6.5</td>
<td>7.1</td>
<td>.9</td>
<td>89</td>
<td>6.0</td>
<td>13.7</td>
<td>.7</td>
<td></td>
</tr>
<tr>
<td>1-10</td>
<td>908 T</td>
<td>21206</td>
<td>.6</td>
<td>6.6</td>
<td>7.2</td>
<td>.8</td>
<td>90</td>
<td>6.6</td>
<td>14.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-11</td>
<td>675 W</td>
<td>14193</td>
<td>1.0</td>
<td>5.9</td>
<td>6.9</td>
<td>11</td>
<td>86</td>
<td>5.9</td>
<td>13.9</td>
<td>.9</td>
<td></td>
</tr>
<tr>
<td>1-12</td>
<td>431 T</td>
<td>9120</td>
<td>.6</td>
<td>6.9</td>
<td>7.5</td>
<td>.5</td>
<td>94</td>
<td>6.9</td>
<td>14.2</td>
<td>.9</td>
<td></td>
</tr>
<tr>
<td>1-13</td>
<td>1260 F</td>
<td>28665</td>
<td>.7</td>
<td>7.1</td>
<td>7.8</td>
<td>.2</td>
<td>98</td>
<td>7.1</td>
<td>15.1</td>
<td>.7</td>
<td></td>
</tr>
</tbody>
</table>

<p>|This Week| 86890 | 3.5 | 33.0 | 36.5 | 3.5 | 91 | 32.5 | 71.1 | 2.5 | 3.5 | 3.7 | 1.9 | 8146 |
|Prev. Wks.| 170403 | 7.3 | 68.1 | 75.4 | 4.6 | 94 | 66.3 | 153.1 | 5.1 | 7.3 | 7.7 | 1.0 | 3.2 | 17503 |
|To Date| 257293 | 10.8 | 101.1 | 111.9 | 8.1 | 93 | 98.8 | 228.2 | 7.6 | 10.8 | 11.4 | 1.0 | 5.1 | 25649 |</p>
<table>
<thead>
<tr>
<th>NAME</th>
<th>CLOCK NO.</th>
<th>TAX CLASSIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S.S. NO.</th>
<th>TEL. NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPT.</th>
<th>(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPT.</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONSTANT DEDUCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUARTER</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>FIRST</td>
</tr>
<tr>
<td>SECOND</td>
</tr>
<tr>
<td>TOTAL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CITIZENSHIP</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EMPLOYMENT RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IN</th>
<th>OUT</th>
<th>REASON</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FIRST</th>
<th>PERIOD ENDING</th>
<th>HOURS</th>
<th>RATE</th>
<th>EARNINGS</th>
<th>DEDUCTIONS</th>
<th>AMOUNT</th>
<th>CHECK NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REG. RATE</td>
<td>OVERTIME</td>
<td>OTHERS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OTH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REGULAR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### NAME

<table>
<thead>
<tr>
<th>CLOCK NO.</th>
<th>TAX CLASSIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M-2</td>
</tr>
</tbody>
</table>

#### REMARKS

<table>
<thead>
<tr>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1966</td>
</tr>
</tbody>
</table>

### ADDRESS

#### B. S. NO.

<table>
<thead>
<tr>
<th>TEL. NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

#### AGE

<table>
<thead>
<tr>
<th>DEPT. (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

#### DEPT. (2)

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>

### CITIZENSHIP

<table>
<thead>
<tr>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

### EMPLOYMENT RECORD

<table>
<thead>
<tr>
<th>IN</th>
<th>OUT</th>
<th>REASON</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QTR.</th>
<th>REG. RATE</th>
<th>HOURS</th>
<th>EARNINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>REG. RATE</td>
<td>OVERTIME</td>
<td>OTHERS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### EARNINGS

<table>
<thead>
<tr>
<th>PERIOD ENDING</th>
<th>HOURS</th>
<th>RATE</th>
<th>REG. RATE</th>
<th>OVERTIME</th>
<th>OTHERS</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### DEDUCTIONS

<table>
<thead>
<tr>
<th>PERIOD ENDING</th>
<th>HOURS</th>
<th>RATE</th>
<th>REG. RATE</th>
<th>OVERTIME</th>
<th>OTHERS</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### AMOUNT OF CHECK

<table>
<thead>
<tr>
<th>CHECK NUMBER</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### W-2
<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>ODD NO.</th>
<th>HOURS WORKED</th>
<th>TOTAL HOURS</th>
<th>HOURS RATE</th>
<th>SCHEDULE HOURS</th>
<th>EARNINGS</th>
<th>DEPARTMENT 1</th>
<th>DEPARTMENT 2</th>
<th>DEPARTMENT 3</th>
<th>OTHERS</th>
<th>TOTAL OF SHEET</th>
<th>SHEET NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Totals**

**Totals from preceding sheet**

**Totals**
# Payroll Register

## Pay Period Ending

<table>
<thead>
<tr>
<th>Employee</th>
<th>Check No</th>
<th>House Worked</th>
<th>Total Hours</th>
<th>Hours Worked</th>
<th>Earnings</th>
<th>Deductions</th>
<th>TOTAL</th>
<th>Earnings</th>
<th>Deductions</th>
<th>Amount Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>JS Quimlan</td>
<td>752</td>
<td>40</td>
<td>810.85</td>
<td>5571.8</td>
<td>-</td>
<td>5332.9</td>
<td>4.1</td>
<td>59.44</td>
<td>52.92</td>
<td>9.52</td>
</tr>
<tr>
<td>J Szlie</td>
<td>756</td>
<td>40</td>
<td>810.85</td>
<td>762.4</td>
<td>-</td>
<td>645.2</td>
<td>10.7</td>
<td>65.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>ML Mann</td>
<td>755</td>
<td>40</td>
<td>810.85</td>
<td>536.8</td>
<td>-</td>
<td>516.2</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>V Hettinger</td>
<td>756</td>
<td>40</td>
<td>810.85</td>
<td>633.9</td>
<td>-</td>
<td>558.4</td>
<td>7.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>R Ferguson</td>
<td>757</td>
<td>40</td>
<td>810.85</td>
<td>333.5</td>
<td>-</td>
<td>299.4</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>Tc Harris</td>
<td>758</td>
<td>40</td>
<td>810.85</td>
<td>598.1</td>
<td>-</td>
<td>554.1</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>J Matthews</td>
<td>759</td>
<td>40</td>
<td>810.85</td>
<td>4952.5</td>
<td>-</td>
<td>4805.6</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>J Burgoson</td>
<td>760</td>
<td>40</td>
<td>810.85</td>
<td>591.3</td>
<td>-</td>
<td>547.4</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>D Terramore</td>
<td>761</td>
<td>40</td>
<td>810.85</td>
<td>524.9</td>
<td>-</td>
<td>482.2</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>M Joyce</td>
<td>762</td>
<td>40</td>
<td>810.85</td>
<td>321.7</td>
<td>-</td>
<td>295.7</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>J Reynolds</td>
<td>763</td>
<td>40</td>
<td>810.85</td>
<td>492.5</td>
<td>-</td>
<td>457.2</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>H Orluck</td>
<td>764</td>
<td>40</td>
<td>810.85</td>
<td>365.6</td>
<td>-</td>
<td>321.6</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>T L Pritchard</td>
<td>765</td>
<td>40</td>
<td>810.85</td>
<td>500.0</td>
<td>-</td>
<td>466.6</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>V Maule</td>
<td>766</td>
<td>40</td>
<td>810.85</td>
<td>491.4</td>
<td>-</td>
<td>457.4</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>S Babachuk</td>
<td>767</td>
<td>40</td>
<td>810.85</td>
<td>592.3</td>
<td>-</td>
<td>548.3</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>R L Sheikh</td>
<td>768</td>
<td>40</td>
<td>810.85</td>
<td>565.9</td>
<td>-</td>
<td>529.8</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>T Trussler</td>
<td>769</td>
<td>40</td>
<td>810.85</td>
<td>525.8</td>
<td>-</td>
<td>481.8</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>D Greff</td>
<td>770</td>
<td>40</td>
<td>810.85</td>
<td>260.0</td>
<td>-</td>
<td>216.0</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>E Study</td>
<td>771</td>
<td>40</td>
<td>810.85</td>
<td>634.8</td>
<td>-</td>
<td>590.8</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
<tr>
<td>E Wagner</td>
<td>772</td>
<td>40</td>
<td>810.85</td>
<td>520.9</td>
<td>-</td>
<td>476.9</td>
<td>10.5</td>
<td>66.54</td>
<td>66.55</td>
<td>-0.51</td>
</tr>
</tbody>
</table>

**TOTALS THIS SHEET**

<table>
<thead>
<tr>
<th>Earnings</th>
<th>Deductions</th>
<th>Amount Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000.00</td>
<td>-</td>
<td>9438.91</td>
</tr>
</tbody>
</table>

**TOTALS FROM PRECEDING SHEET**

<table>
<thead>
<tr>
<th>Earnings</th>
<th>Deductions</th>
<th>Amount Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>7356.38</td>
<td>-</td>
<td>7356.38</td>
</tr>
</tbody>
</table>

**TOTALS**

<table>
<thead>
<tr>
<th>Earnings</th>
<th>Deductions</th>
<th>Amount Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>15356.38</td>
<td>-</td>
<td>15356.38</td>
</tr>
<tr>
<td>Period Ending</td>
<td>Hours</td>
<td>Rate</td>
</tr>
<tr>
<td>---------------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>2-2-68</td>
<td>40</td>
<td>45</td>
</tr>
</tbody>
</table>

The Container Company
COLUMBUS, WASH.

Pay

To the Order of

Eric A Jones

To the National Bank & Trust Co.
of Columbus, Wash.

$129.18

John Norr
Payroll Account No. 2
SYSTEMS FLOWCHART OF WEEKLY PROCEDURE

Factory

- Time Cards
  - Balance
  - File Time Cards

- Production Report
  - Total Employee Hours
  - Master Employee Time Sheet
  - Calculate Gross Pay
  - Post to Payroll Register

- Master Employee Time Sheet

Administrative

- Employee Time Sheets
  - Total Hours

- Balance
  - Post to Labor and Production Report
  - Employee Time Sheets
Calculate Statutory Deductions, Voluntary Deductions and Net Pay

Employee Earnings File

Write Check, Check Stub, and Post to Payroll Register and Earnings Records

Total Each Column of Payroll Register

Total Earnings Records

Crossfoot for Balance

File Payroll Register

Earnings Records

Distribute Checks

Post Earnings Records to General Ledger

File Employee Earnings Records

File General Ledger
Quarterly Procedure

1. Employee Earnings File

2. Total Each Column of Earnings Records

3. Post to Earnings Records

4. Calculate Year-to-Date Totals

5. Post to Earnings Records

6. Crossfoot Earnings Records

7. Type 941A

8. File Earnings Records

9. Write W2 (year-end only)

10. Distribute Copies as Prescribed by Law

11. Crossfoot for Balance

12. Summarize Earnings, FICA, and FIT to 941

13. 941 and 941A Originals to Federal Government

14. 1st Copy 941A

15. File 941A

16. Original 941A

17. 1st Copy 941A Sent to Local Government

18. 2nd Copy 941A

19. Prior 941A
Section 15: SOME PRELIMINARY QUESTIONS AND ANSWERS REGARDING DATA STORAGE

CONTENTS

Introduction ......................... 15.01.00
Data — on Disk or Cards? .............. 15.10.00
   General Considerations ............. 15.10.01
   Flexibility in Order of Processing ... 15.10.10
   Jobs Involving More Than One File.... 15.10.20
   Frequency of Changes to Your File... 15.10.30
   Need for Inquiry into Your File ...... 15.10.40
   Size of Your Data File .............. 15.10.50
   Your Backup Requirements .......... 15.10.60
   Record Size .......................... 15.10.70
   Other Considerations ............... 15.10.80
   Summary ............................. 15.10.90

How to Safeguard Your Disk Data Files .. 15.20.00
   Introduction .......................... 15.20.01
   Know Your Data ....................... 15.20.10
   Know What Can Happen to Your Data .. 15.20.20
   Design an Accident-Insensitive System 15.20.30
   Detect Errors Before They Do
   Damage ............................... 15.20.40
   Plan Modest-Size, Modular Programs 15.20.50
   Always Back Up Your Disk Files with a
   Duplicate Copy ....................... 15.20.60
   Provide Tested and Documented
   Recovery Procedures .................. 15.20.70
INTRODUCTION

Often, before starting the design of a system, there are many questions regarding data storage. Two of the more important are:

- Should I use cards or disks for my data files?
- How can I safeguard my data?

This chapter answers these questions on a broad basis, leaving the details for later chapters.
DATA — ON DISK OR CARDS?

General Considerations

Before you get too far into systems design and programming, you should ask a basic question about every data file you intend to use: Should it be stored on a disk cartridge or in the form of a card deck?

The disk can be an extremely powerful medium for the storage of your data; however, it can be misused. Some data, if placed on the disk, will cause your programmer more work in the long run than if a simple deck-of-cards approach had been used.

In order to lessen the possibility of such a situation, let us answer some of the questions that arise when choosing a storage medium for data.
Flexibility In Order of Processing

In general, your data, whether on disk or cards, contains some master information (names, rates, balances, etc.) in some order or sequence. When you process this information, the transactions may be in another sequence. For example, your employee master data file may be in man-number sequence, while your employee detail cards are grouped by department.

In this situation, the disk has a distinct advantage over cards, since it is a direct access storage device (DASD). This means you can directly access any record, regardless of which record was processed last or which record is next. This allows you complete flexibility in the order of processing.

With your master data on cards, you have to sort both the master deck and the transaction deck into the same order, collate them together, and then process your data in the desired sequence.

Although the disk has a great advantage over cards, its importance varies with the size of the file. Are you talking about 100 employees and a 10-minute sorting job, or 1,000 employees and 45 minutes of card handling? In later sections some other considerations will be discussed that may tip the scales in favor of cards.
Jobs Involving More Than One File

The previous topic can be expanded to consider more than one file, which is the case in many commercial applications. For example, many payroll applications involve a job cost file as well as the employee payroll file. If an employee detail card says that man 607 worked 12.5 hours on job 70976, you can find man 607 in the employee file and add 12.5 hours to his weekly total, then find job number 70976 in the job cost file and add 12.5 hours to its weekly total, all within one program. A card file system would involve:

1. Sorting and collating the employee detail cards with the employee master cards
2. Running the program and punching a new updated employee master card
3. Separating the cards
4. Sorting and collating the employee detail cards again, this time with the job master cards
5. Running a different program, this one punching a new master job cost card
6. Separating the cards and filing them

Depending on the number of cards involved, this could be a cumbersome process. But again, some of the considerations discussed later may override this one.
Frequency of Changes to Your File

A third consideration in deciding on card or disk is the number of times the data in your file must be changed, and the difficulty involved in changing it. Some amount of change is inevitable; in a payroll file every week will bring raises, new dependents, changes of address, etc. These minor changes do not present much of a problem.

With a card file it is very easy; a new card is punched and substituted for the old card.

With a disk file it is somewhat more involved; you must run a change program, which reads the new data from cards or the console keyboard and inserts it in the proper place on the disk record.

Major changes are another matter — new employees, a new group of items in stock, etc. Here again, changing a card file is relatively easy, and changing a disk file more difficult. It is a simple matter to punch a master card for new item number 1705 and place it in the card deck between items 1704 and 1800. It is not quite so simple on the disk, where items 1704 and 1800 are probably adjacent, with no space between them. Either item 1705 is placed in a special area, with a special routine to find it, or the entire file is reorganized, moving every item after 1704 "down" one position to make room for item 1705. This also would require a special program or routine.

If a data file is subject to frequent major (organizational) changes, you may add a few points to the "card file" side of the balance. These points may or may not be enough to swing the decision, since the first two items (processing order and number of files) are more important, and generally favor disk use.

Remember, when you change a field on a card, you still have the old card; when you change some data on the disk (usually an entire record at a time!), the old information is gone. Therefore, special care must be taken to ensure that disk changes are processed correctly the first time.
Need for Inquiry into Your File

In some cases it is very desirable to be able to look into your data file to get certain current information: number of pieces of item number 170653 on hand, year-to-date gross pay of man number 8091, etc. When your data file is in the form of a card deck, this is relatively easy, since you merely find the right card, interpret it, and read the data, much as you would any other hard-copy file — index cards, ledger sheets, etc.

People are accustomed to doing this, and often resist the change to disk-resident files because they cannot "see" what is on the disk.

It is true that data written on the disk is somewhat less tangible than if it were on a deck of cards, but this is not the overriding consideration it is made out to be.

True, it takes a special program or subprogram to read and display data on a disk, so demands for inquiry do add a few points to the "card file" side of the balance. However, a properly designed system can lessen or eliminate these points entirely.

If someone within your company requires, say, the current status of inventory, it may be possible to replace his 5" x 8" card file with a daily listing of stock status, or a weekly listing with daily updates. If he insists on immediate response to up-to-the-minute status, the programmer can build an inquiry subroutine into every program, calling it only when some console switch is turned on:

```
CALL DATSW(7, MM)
GO TO (9,10), MM
9 CALL INQUR
10 CONTINUE
```

These four statements would be placed at a convenient spot in every program. Whenever anyone wanted to inquire of the disk, he would turn on switch 7. The subroutine INQUR would soon be called, and probably request that a part number be entered through the console keyboard. After the requested information was looked up on the disk, it would be typed on the console printer, and the main program would continue.

Large demands for inquiry sometimes make the use of card files appear more attractive than disk files, but proper systems design can often reduce the importance of this factor. In fact, inquiry into a disk-resident file is often a plus factor, since the data obtained would have an up-to-the-minute status.
Size of Your Data File

This item is hard to separate from some of the other considerations. However, all other things being equal, putting very small data files on the disk is sometimes not worth the extra effort, and very large data files will not fit on the disk. Most files fall somewhere in between, and some factor other than size will govern the final card or disk decision.
Your Backup Requirements

Whenever you work with files containing important data (payroll, accounting, etc.), you should not ignore the possibility of accidental destruction of this information. Many accidents can befall card decks — card jams in the reader, floods, spilt coffee, misplacement, etc. Because you can recover from many of these accidents by patching torn cards, duplicating watersoaked cards, etc., it is not too common to find duplicate sets of master card files maintained.

Data files kept on the disk cartridge are subject to a similar list of accidents, but with a difference: it is often impossible to reconstruct the data after an accident, unless you have planned for just such an occurrence.

Because of the need for preplanning, the matter of backup may be considered a disadvantage for the disk file. In actuality, it may be on the plus side, since it forces duplicate files.
Record Size

Because of the physical limitations inherent in a punched card (80 columns), it can be cumbersome to process long records that are kept in card form. Each record may require four or five cards, which must be identified and kept in order. On the other hand, disk records may be as long as 320 words (640 characters). If long records are required, you have a few "plus" points for placing the data file on disk.
Other Considerations

In addition to the factors noted previously, there may be others of equal or greater importance — factors that may be completely unrelated to the particular data file under study. Some typical factors are the storage cost of many cards versus one disk, management's wishes, and the desire to train programmers in disk techniques.
Summary

This section has briefly covered some of the disk-vs-card considerations and attempted to give general guidelines for making this decision. It would be ideal if these factors could be presented in the form of a decision table, score sheet, or other device, but this is not possible. Lacking such a tool, you must study each data file, mull over the pros and cons of disk or cards, and make your own decisions.

Some companies (especially those installing their first data processing system), realizing that their files fall on the borderline, decide to start with card data files. Their reasoning is correct: The system may be less sophisticated and require more machine and operator time, but it is easier to program, use, and understand. Later, if they decide that a certain file should be placed on the disk, it is relatively simple to make the change. The bugs in the system have been ironed out, the programmers are more experienced and confident, and the general atmosphere is more conducive to such a step.
HOW TO SAFEGUARD YOUR DISK DATA FILES

Introduction

This section is of particular interest to those using (or considering) the disk for storage of data files. Accidents will happen, and you must plan ahead to minimize their effect. This is especially true in the case of disk, where data is stored in the form of magnetized spots, recorded at extremely high densities, and read/written by a precise mechanism.

On the other hand, hard copy data is relatively insensitive to accidents. Punched cards can be folded, spindled, and mutilated (even torn, crumpled, splattered with coffee, etc.) without disastrous results. A few minutes (or hours) at the keypunch can remedy all but the most drastic card mishaps. Other paper documents (ledger book, index cards, forms and reports) are not too difficult to duplicate or reconstruct if the original is destroyed.

The purpose of this section, however, is not to discourage the use of the disk for data storage; used properly, the disk offers advantages that overshadow the potential hazards. If you follow the common-sense suggestions in this section, accidents can become rare, improbable events — more a nuisance than a disaster.
Know Your Data

Before starting into a long discussion of how to protect disk data, let us review the various types of data fields in your disk records and determine which, if any, are worth protecting. Naturally, you don't want any of your data lost, but certain items are more important than others, since they are much more difficult to replace.

Take a typical payroll file, where there is a record for each employee:

1. Employee number
2. Name, address, city and state
3. Indicators — marital status, sex, number of dependents, etc.
4. Pay rate
5. Year-to-date dollar figures — gross, taxes, etc.
6. Quarter-to-date dollar figures — gross, taxes, etc.
7. Miscellaneous cumulative — days vacation, sick leave taken, etc.

The first four items are comparatively static, seldom changing, but the latter three probably change every pay period.

If an accident occurs (you should assume the worst possible case), the entire record for every employee is lost. How would you reconstruct your data file? The first four items are easy — the latest information probably exists in the form of a card deck and can simply be reloaded onto the disk. That is how it got there in the first place. However, the last three items present a different picture — they change each pay period. When you write the updated disk record, this week's total is written over last week's total, and last week's total disappears from the disk. Unless you take definite steps to save it before writing on top of it, last week's total will completely cease to exist.

Some disk data fields, therefore, are more critical than others — particularly those that change often, are modified on the basis of previous data (for example, year-to-date gross), or are not kept in duplicate copies.
Know What Can Happen To Your Data

Before you can go about safeguarding a disk data file, you must know what you are safeguarding it against. Basically, there are three general classifications of hazards:

1. **Physical hazards.** Although the disk cartridge is in a sturdy container, it is certainly not immune to careless handling, loss, natural disaster, etc. The cartridge should be stored at moderate temperatures, (between 60 and 90 degrees Fahrenheit) and should not be placed on high shelves or other precarious places. In general, common sense prevails.

2. **Intentional modification.** Payroll data and other confidential information should be kept on disk cartridges dedicated to that use, and should be kept in a secure place. As in the case of physical hazards, there is very little else that can be said about this sensitive area except that common sense must be used.

3. **Accidental modification.** Every program that writes on the disk should be given very close scrutiny. Ask yourself: Is there a chance that wrong information could be written on my disk file? Nine times out of ten the answer is yes. All you need is one mispunched card column, with the resulting wrong answers, and you have a disk record with erroneous data. If the data you are placing on the disk is of a critical nature (as discussed in the preceding pages), you may have problems.

Later sections will discuss some of the ways you may avoid such accidental modification, and how you may easily recover from them. Some of the potential sources of such accidents are:

1. Programming errors (program not completely debugged, etc.)
2. Errors in input data
3. Mistakes by the 1130 operator (running a program twice, etc.)
Design an Accident-Insensitive System

The safety of disk data should be a constant consideration when designing a system. "An ounce of prevention is worth a pound of cure" — or, in data processing terms, a few minutes spent in planning can save many frantic hours or days in keypunching and computer reruns. An accident may never occur, but it would be foolhardy to ignore its possibility. By following a few basic guidelines, the system may be designed so as to be relatively insensitive to accidents; no matter what may happen, recovery is quick and straightforward.
Detect Errors Before They Do Damage

Whenever there is any chance of erroneous data being written on the disk, you should provide a series of checks to minimize the damage. If there is any possibility that input data cards contain bad data, they should be checked. Your keypunch operators should be familiar with the business, so that they can recognize outright mistakes on source documents. Programmers should be urged to build reasonableness checks into their programs.

For example, a program that reads employee labor cards should always check the number of hours worked and, if the number is questionable, take appropriate action (such as type a message and pause).

Program results in the form of printed answers should be spot-checked before the next processing phase. Most errors are easily spotted early in the game, provided someone is there to look for them.
Plan Modest-Size, Modular Programs

At first thought, it would appear that the best program is one that does as much as possible. Why have half a dozen small payroll programs when one could do everything? Unfortunately, however, a large program that does many things tends to compound errors rapidly.

Let us look at the typical payroll job steps for each employee:
1. Read employee's payroll labor card(s)
2. Read his master data from disk
3. Compute gross
4. Compute deductions and net pay
5. Compute all YTD and QTD totals
6. Write his new master disk record
7. Print payroll register
8. Print paycheck
9. Print check register

Suppose you wrote one very large program to do all nine steps and one of the cards for the 56th man somehow got mixed in with the cards of the 108th man. Your programmer has done a good job of error checking, so the 1130 types CARDS MIXED UP and pauses. You have processed 107 employees — printed the register, written checks, updated disk records, etc. — with one man (the 56th) completely wrong. How do you recover?

Correct the cards and rerun from the beginning? No. Besides printing duplicate checks, that would compute and write new YTD and QTD totals for everyone and completely ruin your disk data records.

Keep going and fix the 56th man later? Possibly, but how? This would require a special program to correct his now-erroneous disk record. It would also require a handwritten paycheck, a hand correction to the payroll register, handwritten totals, and a lot of explaining to the accounting department.

Reprogram the entire system to be less accident-prone? Yes, but a little too late. It should never have been written to do so many things.

This example represents an everyday occurrence. Programs are written this way and cause great consternation when the inevitable error in input data occurs — or when the operator enters the wrong week-ending date, or when the paper in the printer jams, etc.

A properly planned payroll system, like the example used throughout the following chapters, would consist of four programs, not one:

- **PAY16**
  - Read Input Cards
  - Check for Errors

- **PAY04, Part 1**
  - Read Input Card
  - Find Man Number on Master Disk File
  - Perform Calculations
  - Update Disk
  - Repeat Steps 1 - 4 for All Employees

- **PAY04, Part 2**
  - When Part 1 is Finished, Print Payroll Register Directly from Master Disk File

- **PAY05**
  - Print Payroll Checks Directly from Master Disk File

- **PAY06**
  - Print Check Register Directly from Master Disk File

The advantages of this plan are obvious:
1. The input cards are checked before they are used to modify the disk records.
2. Payroll checks are printed after the payroll register has been inspected for errors.
Always Back Up Your Disk Files with a Duplicate Copy

Regardless of how the processing system is designed, there should be a duplicate copy of every disk data file. If you have multiple disk drives, you can copy from one disk drive onto another; if you have one drive, you must dump to cards. The copying (or dumping) should be on a regular basis, and should not be left to chance or done whenever there is nothing else to do. Both copying and dumping may be done easily with the Disk Utility Program, as outlined in section 60.

If your 1130 system has only one disk drive, it is impossible to copy disks, and backup must be in the form of cards. Either the DUP *DUMPDATA function may be used, or you may write your own dump program. With large data files, both dumps take a significant amount of time. For example, it takes about three hours to dump a 1000-sector data file.

Because of the time involved, there is a natural tendency to avoid dumping such files. However, an analysis of a typical situation shows this to be self-defeating.

Assume an 800-man employee file, contained in 400 sectors. To dump it with a 1442, Model 6, takes about 60 minutes.

The weekly processing sequence is as suggested earlier:

<table>
<thead>
<tr>
<th>Process</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAY16 Edit</td>
<td>30 min.</td>
</tr>
<tr>
<td>PAY04 Calculations, Disk Update</td>
<td>90 min. and Payroll Register</td>
</tr>
<tr>
<td>PAY05 Payroll Checks</td>
<td>60 min.</td>
</tr>
<tr>
<td>PAY06 Check Register</td>
<td>30 min.</td>
</tr>
</tbody>
</table>

For purposes of analysis, assume the worst possible case — namely, that somehow during PAY04 the payroll data cartridge is completely destroyed. (No matter how improbable or infrequent you think this might be, it can happen.) How do you recover?

If you dump the data file every week, you must:
- Reload the dumped deck 30 minutes
- Rerun PAY16 and PAY04 2 hours

You have completely recovered in 2-1/2 hours.

If you dump every other week, and you again consider the worst case (your last dump was two weeks ago), you must get the data cards from last week, and:
- Reload the dumped deck 30 minutes
- Rerun PAY16 and PAY04 with last week's cards 2 hours
- Rerun PAY16 and PAY04 with this week's cards 2 hours

In 4-1/2 hours you have caught up to where you were before the accident.

If it had been three weeks since your last dump, the reconstruction time would be 6-1/2 hours; four weeks, 8-1/2 hours; five weeks, 10-1/2 hours; etc. Each week adds about 2 hours.

These figures assume that you can immediately lay your hands on the previous week's data cards in the proper order. If this is not so, these times could go up drastically. The figures also assume that everything goes smoothly during the recovery phases. This, however, is not a very safe assumption, since the operators will be rushed and unfamiliar with the procedures.

Without knowing the probability of such an accident, it is impossible to compute the optimum dump frequency. It is probable, however, that you will not want to be in a 10-1/2-hour recovery position, no matter how slight the probability, just to save an hour a week and a few thousand cards.

In this case, the best approach would seem to be a dump every week for the first few months of the installation, every other week after everything has stabilized, and every third week if conditions seem to warrant it.
Provide Tested and Documented Recovery Procedures

It does little good to follow the previous advice if your recovery procedures have not been tested and documented. Usually, time is of the essence, and the operator should not have to study program listings to determine what to do when accidents occur. This is inviting trouble and can turn a minor mistake into a disaster.

If a program checks the input data for errors (as it certainly should), the error messages should be self-explanatory or be keyed to a document that explains exactly what to do. For example, a well written and well documented program will type a message such as ERROR NO. 6 and pause, waiting for the operator to take corrective action. The "run book", or "operation manual", should contain a complete description of what happened and what to do about it. Figure 15.1 shows a typical error recovery sheet; Figure 15.2 is a blank copy of the same form.
<table>
<thead>
<tr>
<th>Section</th>
<th>Subsections</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>20</td>
<td>70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JOB</th>
<th>PROGRAM NAME</th>
<th>PROGRAMMER NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAYROLL</td>
<td>PAY 16</td>
<td>JP</td>
</tr>
</tbody>
</table>

**MESSAGE TYPED:**

ERROR 1 MMMM JJJJ

**PAUSE – DISPLAYED IN ACCUMULATOR**

1 1 1 1

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT**: 176, AND:

CLEARS TOTAL AND GOES TO READ A DETAIL CARD,
ASSUMING IT IS THE FIRST CARD FOR NEW MAN

**DESCRIPTION OF WHAT IS WRONG:**

DETAIL CARD MAN NUMBER JJJJ IS LOWER THAN
LAST CARD, WHICH WAS MMMM. IT SHOULD BE EQUAL
OR HIGHER

**PROBABLE CAUSE:**

THE DETAIL CARD JUST READ IS IN THE WRONG PLACE.
IT BELONGS EARLIER IN THE DECK.

**RECOVERY PROCEDURES:**

REMOVE OUT-OF-SEQUENCE CARDS AND CONTINUE
WITH JOB. HOLD CARDS REMOVED UNTIL PROGRAM
IS RUN AGAIN. ADJUST CONTROL TOTALS AT END
OF RUN.

**COMMENTS:**

MAKE CERTAIN THAT WHOEVER "SORTED" THE
CARDS KNOWS THAT THEY GOOFED!

**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/28</td>
<td>JAH</td>
</tr>
<tr>
<td>3/15</td>
<td>JAH</td>
</tr>
<tr>
<td>IBM 1130 ERROR RECOVERY SHEET</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td></td>
</tr>
<tr>
<td>JOB ________________________</td>
<td></td>
</tr>
<tr>
<td>PROGRAM NAME ________________</td>
<td></td>
</tr>
<tr>
<td>PROGRAMMER NAME ____________</td>
<td></td>
</tr>
<tr>
<td>MESSAGE TYPED: ______________</td>
<td></td>
</tr>
<tr>
<td>PAUSE – DISPLAYED IN ACCUMULATOR</td>
<td></td>
</tr>
<tr>
<td>AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT ______________________</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION OF WHAT IS WRONG: ________________________________</td>
<td></td>
</tr>
<tr>
<td>PROBABLE CAUSE: __________________________</td>
<td></td>
</tr>
<tr>
<td>RECOVERY PROCEDURES: __________________________</td>
<td></td>
</tr>
<tr>
<td>COMMENTS: __________________________</td>
<td></td>
</tr>
<tr>
<td>SCORESHEET DATE</td>
<td>INITIALS</td>
</tr>
</tbody>
</table>

Figure 15.2.
Section 20: 1130 APPLICATION DESIGN

CONTENTS

Introduction ........................................ 20,01.00
Accounting Controls ............................. 20,10.00
   Review of Accounting Control
   Principles .................................. 20,10.10
   More Specific Suggestions for Document and Accounting Controls ...... 20,10.20
Form Design .................................... 20,20.00
   1130 Considerations ....................... 20,20.10
   Form Design Principles ................. 20,20.20
Card Design .................................. 20,30.00
   1130 Considerations ....................... 20,30.10
   Card Design Principles ................. 20,30.20
Design of Disk Data Files ...................... 20,40.00
   Introduction ................................ 20,40.01
   Data ...................................... 20,40.10
   Field Size .................................. 20,40.20
   Data Sequence ............................. 20,40.30
   File Organization .......................... 20,40.40
   Record Format and Blocking ........... 20,40.50
   File Processing ............................ 20,40.60
   File Control ................................ 20,40.70
   Payroll Example ........................... 20,50.00
   Narrative .................................. 20,50.10
   Card Forms and Console
   Keyboard Input .............................. 20,50.20
   Console Printer and Line Printer
   Forms for Output ............................ 20,50.30
   Disk Record Formats ........................ 20,50.40
   System Flowchart ........................... 20,50.50
   Language Selection .......................... 20,60.00
   Introduction ................................ 20,60.01
   Programming Languages ................. 20,60.10
   Application Programs ...................... 20,60.20
   Which Programming Language or Application Program Should You Use ? .......... 20,60.30
INTRODUCTION

Up to this point, you have been concerned mainly with planning and gathering facts about the way you are processing your data now. The next step is to take this information and mold it into application designs for the 1130. Follow this plan:

1. Be sure your current-system documentation is complete. This cannot be overemphasized, because time gained by doing a hasty but poor job in documentation will be lost later. In fact, it will probably be lost several times over, because of the need to sift out errors of omission from other design and programming errors, research the omitted facts, then rewrite and retest all affected programs.

2. Design or redesign your reports. This must be done in detail, and all interested parties should sign off on the new layouts. The forms layouts illustrated later in this section are sufficient to guide data processing personnel in their programming. The people who will use these forms should be shown samples, as they will actually appear, with real data hand-printed in the formats that are planned.

3. Lay out the cards (see 20.30.00).

4. Draw flowcharts of the procedures you will follow in processing the cards to produce the reports. Decide what programming system or application program (such as 1130 Commercial Sub-routine Package) you will use for each run, and note it on your flowcharts.

5. Establish procedures for accounting controls where you need them. They may be different from those you are using now.

(Steps 2-5 are usually overlapped to a large extent. Changes in reports usually require changes in card formats, procedures, and accounting controls. Similarly, changes in any one of the latter three elements affect the others.)

6. Hold a management review after the first application has gone through the steps above.

7. Let each person who is programming carry one of the programs in this initial application through Program Development (see section 25) to completion. By so doing, he will broaden his experience quickly, develop a more realistic idea of the amount of time required for application design, programming, and testing, and get a clearer idea of the depth of detail needed in your current system documentation. After this experience has been gained, review your activity schedule dates and adjust them according to what you have learned.

This section reviews the important considerations of designing accounting controls, forms, and cards. Then the same payroll example introduced in 10.50.00 is presented, along with typical form and card designs and job flowcharts, as well as the disk record layouts for the programs required to do this payroll.

To help you decide which language to use for any given run, this section also covers the considerations that go into language selection. Finally, it presents an example of a method for estimating the length of time required to run a program.
ACCOUNTING CONTROLS

This section covers the subject of accounting controls at two levels. The first part, "Review of Accounting Control Principles", points out the types of control that are required and serves as a review if you have set up and used controls before. It ends with a short summary.

The second part, "More Specific Suggestions for Document and Accounting Controls", deals with specific examples of control sheets and methods of control, and can be used as source material for setting up your own control documents and procedures.
Review of Accounting Control Principles

Accounting controls are an essential part of your data processing installation. The inherent accuracy of data processing equipment and the elimination of many error-prone clerical steps helps reduce the number of errors in processing; however, good accounting practice requires that you have a precise procedure available to prove and reconstruct the basic records of the system. Controls are also necessary to guard the records of your business against fraudulent acts.

The accompanying exhibit shows a typical information flow through a system. Information from source documents is punched into cards. The first control (A1) ensures that your information was transcribed correctly and completely. This can be determined in one of several ways:

1. The cards can be key-verified.
2. Control totals from the source documents can be balanced against the card totals. For example, an adding machine total of the quantities on a batch (several source documents) can be balanced against the total of the quantities in the cards created from the documents. This same technique can be used to control other numeric fields, such as employee number, part number, etc. The total is often called a "hash" total. If an out-of-balance condition occurs, a listing of the cards provides a means of comparing the card information with the source documents, and the error is quickly isolated and corrected.
3. Document or transaction counts can be used to ensure that a card document is created for every transaction.

Since the information in the cards will be used to update several associated records, accuracy is of prime importance.

At the time the cards are run through the system for accumulating control or hash totals, other tests can be performed.

Fields may be tested for size or reasonableness. For example, the nature of your business may be such that the quantities have reasonable limits, based on the class of an item. Any item exceeding the limit can be so noted for review before processing. This type of test might keep you, for instance, from shipping 24 bathtubs to a small country store as a result of mispunching an item number for pliers.

Batch size (the number of documents included in a control group) should be small enough to keep error tracing from becoming too cumbersome. On the other hand, it is not reasonable to control each document separately.

As the documents enter the process (A2), the same control list above can be used on a cumulative basis to ensure that all the cards from the several batches that constitute a processing run are completely processed.

Card documents being created by the process can be balanced against your control (A3) totals.

A control should be maintained on all card files (A4). The total from (A3) will be used to update this control as cards enter the file. Before processing a large card file, it is often advisable to run a trial balance on the file--particularly if the file is being updated or used as a source of reference between processing runs. The purpose of this trial balance is to catch errors in filing, missing cards, duplicate cards where a change was made but the old record was not removed, etc.

A control listing of all cards entering and leaving the file establishes the control total entry and provides an audit trail if it is necessary to identify lost or duplicate records.

The accompanying exhibit also shows an example of a simple control sheet. Each time records enter the file or are removed, an appropriate entry is made and the file balance is updated.

It is often possible to provide audit requirements as a by-product of creating normal reports. For example, the trial balance of your file might be a stock status report. The value of separate balancing runs must be determined by experience for each application and will vary with the experience and capability of your operating personnel.

The number and types of controls will depend a great deal on the application. Your own auditors should be consulted in determining control procedures. Controls and audit trails should conform with their requirements and should be established as an integral part of the data processing procedure. Much of the material in subsection 20.10.20 will help you and your auditors design adequate control forms.

In setting up controls that will operate successfully, the following should be kept in mind:

1. Only those controls that satisfy a need should be included.
2. The overall system of controls should be conceived and arranged for when procedures are being planned. Thus they will be an integral part of each procedure, and those areas that may have a tendency to be overcontrolled or undercontrolled will be spotted.
Typical control of data processing system
3. Personnel who maintain the controls should be familiar with machine functions so that they can locate, determine the cause of, and correct out-of-balance conditions.

4. Controls should be simple and easy to maintain so that workflow is not disrupted.

5. A description of control operations should be documented and assembled for reference and training purposes.

6. Whenever possible, control operations should be mechanized.

7. When documents to be processed are batched, batch size should be such that work will continue to flow steadily.

8. Company auditors should agree with the audit and control procedures.

9. Department controls should always be established outside the department, at the source of the data.
More Specific Suggestions for Document and Accounting Controls

The following discussion of accounting controls is concerned with (1) those controls established and used outside the data processing installation and (2) those established and used inside the installation. Outside controls consist primarily of the initiation, authorization, and verification of source documents representing accounting transactions. Inside controls consist of (1) checking operations, in which transcribed transaction data is verified, and (2) balancing operations, which ensure the accurate processing of all transaction data.

Generally, the necessity for accounting control increases with the volume of transactions or documents processed and the complexity of operations performed. A variety of control techniques will be discussed. The techniques to be employed depend upon individual conditions within your organization. It is important that the controls which you use always provide a proper balance between their cost and their value. Since a system of accounting controls may be obsoleted by a change in accounting procedure, company policy, company organization and/or data processing equipment, controls should be examined and evaluated periodically. Company auditors should participate in establishment and evaluation of a control system.

Outside Controls

Control techniques described in the following text are not necessarily limited in use to any one application; they are easily modified for use with a variety of applications.

Document register. Control of individual documents can be maintained effectively by the preparation of a register on which each document is listed at the point of receipt or origin. The register should include either a description that is sufficient to identify each document quickly, or a serial identification number. The serial number not only furnishes positive identification and an effective method for later reference, but is also most easily used at the point of entry or origin. When each document has been completely processed, it is "checked off" or canceled on the register. Uncanceled numbers represent documents that either are in process or have been mislaid. Intermediate processing operations for each document may be shown on the register and dated as the document passes those points in the procedure. The illustrated document register for sales orders not only discloses a missing or misplaced document, but also indicates any delays in processing—as might be the case with order 12843, which, several days after its receipt, has not yet been billed.

Serial numbering and the batch control ticket. Where serial numbers are printed or stamped on each document, rearrangement in serial number order and a check for missing numbers may be performed during, as well as after, processing to ensure inclusion of all documents. This plan is particularly adaptable to documents such as checks or drafts, where each document must be accounted for. When the document is an IBM card, the serial number may be punched into, as well as printed or interpreted on, the card; arrangement of the documents, as well as a count of and sequence check for missing documents, may then be accomplished automatically.

Serial numbering may also be used for groups and batches. If so, the number of documents in each batch is recorded, together with the batch serial number, either on the first document or on a separate form accompanying the batch. For large-volume operations, batch size should be predetermined for ease and efficiency in handling.

The illustrated batch control ticket employs a document count as well as document and batch serial numbering. By maintaining a file of the batch control tickets, both the sending and receiving departments can account for all documents.

Transmittal and route slips. A letter of transmittal describing a group or batch of documents is frequently employed to establish control and transfer responsibility when documents move from one department or location to another. The transmittal slip, as shown, is usually a printed form with spaces to indicate the variable information for the batch.

When the volume of work or the number of people who may perform any given operation is large, it may be desirable to fix responsibility for accounting for documents passed from each operation to the next as well as from one department to another. In this case, a route slip is employed, either in addition to or in combination with the letter of transmittal. The route slip is similar to the batch control ticket shown, except that in this case each department, or operational step performed, is identified, along with an indication of the processing time and the operator or clerk responsible for each job. Responsibility is fixed, and the means to effect a degree of work control as well as document control has been incorporated into the same form.
### ORDER REGISTER

<table>
<thead>
<tr>
<th>Date Received</th>
<th>Order Number</th>
<th>Date Audited</th>
<th>Date Billed</th>
<th>Date Shipped</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/12</td>
<td>12831</td>
<td>10/14</td>
<td>10/12</td>
<td>10/18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12832</td>
<td></td>
<td>10/16</td>
<td>10/16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12833</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12834</td>
<td>10/15</td>
<td>10/17</td>
<td>10/19</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12835</td>
<td>10/15</td>
<td>10/17</td>
<td>10/17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12836</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12837</td>
<td>10/15</td>
<td>10/17</td>
<td>10/19</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12838</td>
<td>10/15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12839</td>
<td>10/15</td>
<td>10/17</td>
<td>10/17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12840</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12841</td>
<td>10/15</td>
<td>10/17</td>
<td>10/17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12842</td>
<td>10/15</td>
<td>10/17</td>
<td>10/18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12843</td>
<td>10/15</td>
<td>10/17</td>
<td>10/18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12844</td>
<td>10/15</td>
<td>10/17</td>
<td>10/19</td>
<td></td>
</tr>
</tbody>
</table>

**Order register**

### Batch Control Ticket

<table>
<thead>
<tr>
<th>Batch No.</th>
<th>Date</th>
<th>No. of Documents</th>
<th>Received Attached Documents Specified Above</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>10/15</td>
<td>37</td>
<td>12355, 12391</td>
</tr>
</tbody>
</table>

**Batch control ticket**

### Card Shipment Transmittal

<table>
<thead>
<tr>
<th>Date</th>
<th>Destination</th>
<th>Number of Cards</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/16</td>
<td>New York</td>
<td>3000 cards</td>
<td>$25,643.21</td>
</tr>
</tbody>
</table>

**Transmittal and route slips**
Cancellation and time stamps. As a document is received at a control point or passed through a given department, it may be "canceled" by a stamp to indicate that it has reached or passed through a certain stage in its processing. Any clerk or operator handling documents would automatically reject or return for checking any document not bearing the correct cancellation. As shown in the accompanying illustration, the use of the time stamp for cancellation affords, in addition to document control, a method of achieving work time or production control, since it furnishes an accurate, unalterable record of elapsed time for handling.

Matching. The reassembly and matching of duplicate documents can be used to effect control. This technique is particularly useful when multiple copies are prepared, as with carbon copies, and each copy is then used to prepare records at a different location, for example, purchasing department and receiving department. When all copies are reassembled and matched at the predetermined point, the presence of all copies indicates complete processing. If the documents are punched cards, matching and checking can be done automatically.

Control of factors subject to change. Factors used for calculations and processing may be reviewed and changed from time to time. Examples of such factors are discounts, selling prices, credit limits, commission percentages, and inventory reorder levels.

Controls must be established that allow only authorized changes to be made. This is accomplished by requiring a signature with each request for change (see change authorization exhibit). Changes are documented by printing a register (see change register exhibit). A copy of the report is routed back to the initiating department for review and approval.

Inside Controls

Controls within the data processing installation should ensure that all transactions are processed completely and accurately. The series of checks and balances that make up these controls must begin with the entry of transactions into the data processing installation and continue throughout processing.

---

**PURCHASE ORDER**

NEW MEXICO COMPANY

HOUSTON, TEXAS

DATE 10/12

TO

GENERAL MANUFACTURING COMPANY

ENDICOTT, N. Y.

<table>
<thead>
<tr>
<th>ITEM</th>
<th>DESCRIPTION</th>
<th>QUANTITY</th>
<th>DESCRIPTION</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQUARE SHEAR SWIVEL</td>
<td>40</td>
<td>112002</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>PLY TOP KNEE</td>
<td>10</td>
<td>11302</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>S/S SHEAR W/ BSK</td>
<td>10</td>
<td>12703</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>BOLT AND NUT SHAVE</td>
<td>10</td>
<td>21015</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>RND SHEKING STEM</td>
<td>10</td>
<td>64106</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>BOLT AND NUT SHEAVE</td>
<td>10</td>
<td>66110</td>
<td></td>
</tr>
</tbody>
</table>

**RECEIVING DEPARTMENT**

OCT 14 10:47 AM 19-

W.G.生态环境

---

Time stamp cancellation
The following price changes should be made:

<table>
<thead>
<tr>
<th>ITEM NO.</th>
<th>DESCRIPTION</th>
<th>NEW PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 2685</td>
<td>PEA SOUP $</td>
<td>6.001</td>
</tr>
<tr>
<td>12 3074</td>
<td>ORANGE JUICE</td>
<td>3.857</td>
</tr>
<tr>
<td>13 1111</td>
<td>HAND SOAP</td>
<td>2.200</td>
</tr>
<tr>
<td>13 2954</td>
<td>CONDENSED MILK</td>
<td>1.639</td>
</tr>
<tr>
<td>13 4182</td>
<td>TOOTH PICKS</td>
<td>.353</td>
</tr>
</tbody>
</table>

Change authorizations

CHANGE REGISTER

<table>
<thead>
<tr>
<th>DATE</th>
<th>ITEM CODE</th>
<th>DESCRIPTION</th>
<th>FACTOR BEFORE CHANGE</th>
<th>FACTOR AFTER CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>11-26</td>
<td>12 2685</td>
<td>PEA SOUP</td>
<td>5.956</td>
<td>6.001</td>
</tr>
<tr>
<td></td>
<td>12 3074</td>
<td>ORANGE JUICE</td>
<td>3.132</td>
<td>3.857</td>
</tr>
<tr>
<td></td>
<td>13 1111</td>
<td>HAND SOAP</td>
<td>2.253</td>
<td>2.200</td>
</tr>
<tr>
<td></td>
<td>13 2954</td>
<td>CONDENSED MILK</td>
<td>1.652</td>
<td>1.639</td>
</tr>
<tr>
<td></td>
<td>13 4182</td>
<td>TOOTH PICKS</td>
<td>.352</td>
<td>.353</td>
</tr>
</tbody>
</table>

Change register
Control techniques and devices. A list of control devices and techniques, many of which can be incorporated into the procedure for any data processing system, includes:

- Serial numbering. The serial numbering of orders, invoices, checks, etc., provides control while the data is in transit. Each item or document in the series or group is assigned a successive number; an indication of the beginning and ending numbers accompanies the group.
- Batching with a document or item count. In batching data with a document or an item count, the items or documents are counted instead of numbered; an indication of the count accompanies the group. This technique can be used to control data, both before and after it is punched into cards—for example, requisitions, changes, receiving reports, and punched cards for various analysis reports.
- Batching with a control total. In batching with a control total, some data field that is common to all items or documents is accumulated for the control total, which then becomes the basis for balancing operations during processing. The control field may be an amount, a quantity, an item code, an account number, etc.; totals based on an account number or code are known as "hash" totals. An advantage of this technique is that balancing can often be performed during regular machine processing operations at no extra cost in time.
- Crossfooting. Crossfooting is the addition and/or subtraction of factors in a horizontal spread to prove processing accuracy. It can be used on a payroll register to prove that the final totals of net pay and deductions equal the final total earnings; this provides control on report preparation as well as calculating and card-punching operations. In posting transactions to records that are temporarily stored in a computer (for example, accounts receivable), crossfooting is used to prove the accuracy of posting, either as each transaction is posted, or collectively at the end of the run, or both.
- Zero balancing. Zero balancing is an effective method of verification when both detail items (for example, accounts payable distribution cards or records) and their summary (for example, an accounts payable disbursement card or record) are processed together. Each detail item is accumulated minus, and the summary plus. The result is a zero balance if both are correct.
- Negative balance test. It is possible to detect a change in sign during arithmetic operations and either stop the machine or signal the condition for subsequent review. In payroll applications the sign check is used to indicate the condition in which deductions exceed gross pay; in accounts receivable, accounts payable, inventory, and general ledger applications it can be used to recognize any balance that becomes negative.
- Blank field test. This means checking any data field for all blank positions. As a computer control, it can be used to prevent the destruction of existing records in storage, indicate when the last item from a spread card has been processed, skip calculation if a rate or factor field is blank, etc.
- Comparing. Comparing, as a control technique, permits data fields to be machine-checked against each other to prove the accuracy of matching, merging, coding, balancing, reproducing, gang punching, and record selection.
- Sequence checking. Sequence checking is used to prove that a set of data is arranged in either ascending or descending order before it is processed. It is generally a mechanized operation and may be performed in a separate machine run or simultaneously with another operation in one run.
- Visual comparisons. Comparisons are based primarily upon experience, past performance, and a knowledge of trends that have intervened. By knowing the status as of a certain time and observing trends since that time, it is possible to determine to some degree whether present records represent a complete and accurate picture. For example, present-period payroll is often compared against last-period payroll to spot any questionable variations.

Controls on processing operations. The number of available techniques indicates the need for a thorough study of your application and equipment in order to come up with a system of controls which is adequate but which does not overcontrol and delay processing. In so doing, it is desirable to mechanize as many controls as possible. Mechanized controls are always performed at a constant, rapid speed; manual ones are not. A study of the application will reveal:

1. How closely it is to be controlled.
2. Points in the procedure at which controls must be placed.
3. The correcting and restart procedures to be employed at each point, in case the operation does not balance. If the procedure is a manual one, it should be clearly documented for operator reference and training purposes.
4. How accounting control responsibilities are to be divided.

The basis for control during processing must be established as data enters your installation. This
is generally done when transactions are edited and may consist of assigning a system of serial numbers or developing a document count, a transaction count, an item count, a tape listing and total of some field such as quantity, amount, or code, etc., or a combination of these. When these preliminaries are taken care of, your transactions are ready for processing.

During report preparation, the primary control objective is to prove that all items (accounts or transactions, etc.) are included in the processing and that arithmetic is performed accurately. It can be assumed that the data itself is correct, since punching, summary, and posting operations are proved as they occur.

To ensure the inclusion of all items in the report, a final control total is developed during processing and balanced at the end of the run to a predetermined one. In cycle billing operations, the control may be an account number hash total of those accounts that are in the cycle; for other reporting operations it may be a control total based upon an amount, a quantity, or another code field. For control of arithmetic functions that occur during report preparation, the following techniques should be investigated: crossfooting, total transfer, zero balancing, parallel balancing, reasonableness test, or a combination of these.

Built-in checks and controls. Built-in checks should be taken advantage of and not duplicated by programmed or manual controls. They function as a result of internal machine circuitry and are therefore performed automatically. For example, all machines have checks which stop the machine for an operation that is impossible or in conflict with another.

Computers utilize input/output checks. The input check ensures that all data is read and coded correctly. The output check ensures that your output characters are correctly set up for punching and/or printing.

This discussion does not include all built-in checks; for more information regarding a specific piece of equipment, refer to the reference manual describing the machine.

The audit trail. An audit trail must be incorporated into every procedure; you should provide for it early so that it becomes an integral part. In creating an audit trail it is necessary to provide:

1. Transaction documentation that is detailed enough to permit the association of any transaction with its original source document.

2. A system of accounting controls which proves that all transactions have been processed and that accounting records are in balance.

3. Documentation from which any transaction can easily be recreated and its processing continued, should that transaction be misplaced or destroyed at some point in the procedure.

The audit trail shown in the accompanying exhibit might be found in an accounts receivable application. The original or entry sales register is prepared by date of entry immediately after the cards have been punched or activated from a file. All punched information is listed on the register in detail, so that if a transaction has to be recreated at some later time, reference to the source document will not be necessary. To prove the accuracy of the register's preparation, its final total is balanced to a predetermined one; if the two are equal, the final total is also posted to the control sheet. The sum of these individual totals on the control sheet establishes the final control total to which all accounts receivable operations for the period must balance.

Cards for the cash receipts book are either punched or activated from a holding file. After being prepared in detail, the cash receipts book is balanced to a predetermined total. If it is in balance, the final total is posted to the control sheet and the receipts are posted to accounts receivable.

When the aged trial balance is run, the final total should equal the difference between total debits and total credits to accounts receivable; this difference is available from the control sheet. If the totals do not agree, all the transactions for the accounting period can be sorted into entry date sequence, summarized, and checked against the daily entry totals on the control sheet to isolate the entry date that is out of balance. The transactions for that date are relisted; an entry-by-entry comparison on the duplicate and original entry registers will reveal the discrepancy so that a correcting entry can be initiated.

The sales register and cash receipts book provide documentation that is sufficient for reconstructing a transaction or associating it with the original source document. Balancing each register to a predetermined total proves that all transactions in the group have been processed through that point. The entries on your control sheet provide the means for balancing accounting records at the end of the accounting period.
Audit trial

Incoming transactions listed in detail for permanent audit reference

Totals are posted to the control sheet from both daily registers

Check to Trial Balance
FORM DESIGN

The first part of this section, written for persons familiar with punched card processing, deals with 1130 considerations only.

The second portion is more detailed and serves as an introduction to the subject for those who are new to automated data processing.
1130 Considerations

1. The ability of both FORTRAN and the 1130 Commercial Subroutine Package to provide heading information can greatly reduce the cost of forms. Standard stock forms can be used for all internal reports, and appropriate headings can be provided at the time the report is prepared. Setup time can be significantly reduced by eliminating the need to change forms in the printer.

2. The 120-character print line is probably at least equal to your present capacity. Consider printing narrow forms two-up—that is, two pages side by side (on special paper for splitting), printed at the same time. This technique can double your output or can avoid the need for extra runs or extra carbon copies where the number of required copies is large.

3. The extra speed of your printer (1403) may allow you to make some short runs twice instead of buying expensive multiple-part paper just for those runs.

4. Interchangeable chain cartridges for the 1403 allow you to improve the appearance of certain reports by providing a variety of special characters. Also, printing speed can be considerably improved by selecting a character set containing only the characters you need.

5. The ability to have both the 1403 and 1132 on line can save time, in some cases, by eliminating the need for rerunning cards to produce a second, different report.
Form Design Principles

The design of effective, economical forms requires a certain amount of preparatory evaluation and analysis. The major objectives are legibility, simplicity, economy, and efficient preparation. Local IBM representatives should be consulted early; their guidance and reference materials may help avoid costly mistakes. Steps to be taken in forms design are:

1. Establish the need for the new form. Similar forms may exist which, with minor changes, will satisfy the new requirements.
2. Study the machine to be used for printing. In so doing, use the reference manual for that machine; most manuals have at least one section devoted to the tape-controlled carriage and/or form design. These sections contain valuable information about forms specifications as well as different printer characteristics and operation.
3. List all types of information to be recorded and the number of positions that will be allotted for printing each. In doing this, assemble and study past and present statistics. These can be evaluated in light of future plans and then used as an indication of probable needs. One of the greatest weaknesses in forms design is the tendency to burden a form with unnecessary information. Since entire data processing procedures may be geared to the preparation of a certain report, extraneous information can be costly.
4. Lay out the form on a printer spacing chart. In using the spacing chart the following tips will be helpful (some will be discussed in greater detail later):
   - Use bold type to make special information or headings stand out.
   - In columns for figures allow sufficient space for the largest amount plus punctuation.
   - Place filing information near the top of the form.
   - Title the form.
   - Include form number, date, and page number.
   - Keep headings small, to allow room for written data.
   - Consider total headings at the bottom of the form.
   - Use different-colored copies as an aid in routing.
   - Use double-ruled lines to set off sections.
   - Avoid horizontal rulings as much as possible.
   - Consider guide marks for names, addresses and folding.
5. Make a test using a copy of the proposed form. Examine the report carefully to make certain that zeros are printing properly and that amount fields are large enough.

During the creation of a form the designer should understand and keep in mind the following:

Form width. The overall width of a form is important in determining printing space. Although the IBM form-feeding devices available will handle a great variety of document sizes, certain practical aspects should be observed.

Form costs can be reduced by confining widths to the standard sizes of paper stock used by business forms companies. (These sizes can obtained from the companies; reference to the individual machine manual will indicate which are acceptable.)

In addition, width standardization permits (1) purchase of report binding and filing supplies in fewer sizes and greater quantities at reduced cost, (2) more convenient forms handling, and (3) a reduction in the setup time of form-feeding devices.

Form length. The total number of body lines in a form (regardless of whether six-or eight-lines-per-inch spacing is employed) can be any whole number, up to 132. It should be evenly divisible by two in the case of double spacing, and by three in the case of triple spacing.

Horizontal spacing. All printing is ten characters per inch. Vertical lines separating fields and decimal positions should be drawn so that each splits a printing position. If they are drawn between adjacent positions, paper shrinkage, variations in form insertion and alignment, as well as other variables, may prevent satisfactory registration during printing.

Vertical spacing. The vertical spacing of the printers is under operator control and can be set for six or eight lines per inch. The importance of this is that double spacing at eight lines per inch permits 33-1/3% more lines to be listed on a page than double spacing at six lines per inch. While it is true that six lines per inch at single spacing gives more items than eight lines per inch at double spacing, the latter offers increased legibility.

Form skipping. The maximum distance that can be skipped without losing machine time is not the same for all printers. The individual machine or systems reference manual should be read so that little or no processing time is lost.
Form alignment guide. If possible, a guide for form alignment should be determined and preprinted on each form to facilitate machine setup operations. It is important that a description of the form alignment guide and its use be included in the operation manuals. A delay in machine setup will create a delay in processing.

Numeric amounts. In determining the number of print positions needed for numeric fields, the size of the total must be provided for, rather than the size of the detail amounts. If marks of punctuation are to be machine-printed, the size of the field should be checked to make certain that printing positions have been allotted.

Printable characters. The standard printable characters are:

A – Z
0 – 9
+ / ( - $ , . & *

More information may be found in the appropriate machine reference manual.

Marginal perforations. Most forms have a vertical perforation 1/2" from each side. Sometimes, however, forms are designed with dissimilar margin widths. For example, a form with an overall width of 9-7/8" may be perforated 1/2" on the left and 7/8" on the right, to leave an 8-1/2" x 11" letter-size report after the marginal strips are removed. Many such variations in margin size are used. At least one unused printing position should be left between a machine-printed character and a perforation.

Since some report binders utilize the form-feeding holes for binding, many reports are set up with no perforation on the binding side. The practice of eliminating perforations and letting the form-feeding holes remain on both sides of the finished reports is being followed more and more, particularly with internal reports.

Binding. In most cases, it is desirable to minimize binding space in order to reduce form cost. Therefore, information that will be referred to least should be placed nearest the margin, since it becomes more difficult to read information near the binder posts as sheets are added to a binder.

Because of the amount of space required for headings, many forms can be bound at the top, with no sacrifice in readability. If it is desirable to bind continuous forms without bursting them or binding them on the side, binding holes can be punched in both the top and bottom of the forms.

Carbon copies. Substantial savings can be realized by minimizing the number of carbon copies. Some techniques that are effective in doing this are:
1. Side-by-side duplicate reports
2. Consolidation of reports for multiple use
3. Sequence-routing of reports to different departments, instead of simultaneous distribution
4. Mechanical or photographic reproduction

Any report that is subject to constant usage, such as a weekly timesheet, should be prepared on a durable grade of paper. For most multiple-copy work, the first, or original copy and the last copy are heavier in weight than the intermediate copies. Lighter weights of paper have less cushioning effect on the printing impact, and therefore permit more legible printing on multiple copies. On the other hand, the paper must be of sufficient weight and strength to prevent tearing while feeding or ejecting forms.

The carbon paper used should produce the required number of legible copies without excessive smudging. Various carbon forms in use include:
1. One-time carbon. This is used once and discarded.
2. Carbon-backing paper. The carbon surface is on all or part of the reverse side of the original.
3. Chemical-coated paper. The chemical coating on the back on one sheet reacts with the coating on the face of the next, under the impact of the printing mechanism.

Type style is also an important consideration for multiple carbon copies. Standard type gives maximum legibility. A smaller type style tends to "fill in" when printed through several sheets of paper; with a bolder type style the force of the hammer blow is spread so that sharpness and density are decreased.

The legibility of some special-purpose type is limited. Since it is fixed in size, the more characters that are crowded within the area, the smaller each character becomes. Therefore, as the number of carbon copies increases, the definitive lines of each character seem to become broader. The result is a character that is difficult to read.

In some cases carbon paper is narrower than the form. It may be held in place by a fastening technique at the horizontal perforations between forms, or by some other method such as stitching, gluing, or marginal perforations.
The recommended maximum distances between fastenings are:

<table>
<thead>
<tr>
<th>Form Length</th>
<th>Maximum Distance between Fastenings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 5 inches</td>
<td>5 inches</td>
</tr>
<tr>
<td>5-1/2 to 11 inches</td>
<td>11 inches</td>
</tr>
<tr>
<td>11 to 14 inches</td>
<td>7 inches</td>
</tr>
<tr>
<td>14 to 17 inches</td>
<td>8-1/2 inches</td>
</tr>
</tbody>
</table>

For forms more than 17" in length, the maximum distance between stitches should be determined by actual test.

If staples are used, they must:
- Be located out of the printing area.
- Be properly crimped so they won't catch on guide edges or staples in succeeding forms.
- Not cause excessive bulging during feeding, particularly at the out-fold.

Form types. Depending on its purpose and destination, the form on which a report is printed may range from the least expensive blank stock to custom design. Imprinted stock forms are standard-size forms which are stocked in large quantities and upon which lines, headings, markings and some designs are printed as desired. Custom forms are those designed to fill special needs of size, complexity, and design. Although more expensive, they can be used advantageously to "sell" your company.
CARD DESIGN

This section is divided into two parts. The first provides information that will be useful to a person who has had punched card experience but wants to become familiar with the considerations unique to the 1130. The second deals with more basic card design principles. A more extensive coverage of the subject is contained in the IBM manual Form and Card Design (C20-8078).
1130 Considerations

1. Lining up similar fields between cards is desirable for ease of recognition, for offline punched card processing, and for ease of card handling. A program can as easily define a field in one set of columns as in another.

2. The results of calculations often do not have to be punched into cards. It takes but a few milliseconds for the computer to recalculate the same figure the next time it needs it.

3. The EBCDIC character set contains 256 possible codes. However, many of them cannot be handled by standard FORTRAN programs. Only 53 characters are permitted in card input (see the FORTRAN manual, C26-3715); of these, only 48 may be printed by the 1132 Printer.

4. Normally, an 11-punch over the units position of a field indicates to the 1130 Commercial Subroutine Package that a field is negative, while a 12-punch or no-zone indicates that it is positive. The combinations (11-0) and (12-0) are not valid FORTRAN codes. However, the 1130 Commercial Subroutine Overlapped I/O Package can handle them.

5. Punching speed for serial punches (1442) varies with the last column punched. For example, if the card is to be punched in cc 1-10, 176 cards per minute can be punched on a 1442, Model 6. The same data can be punched in cc 71-80 at only 49 cards per minute. Therefore, fields to be punched should be placed close to column 1. Fields to be read can then be placed anywhere to the right of fields to be punched, with no effect on card reading speed.
Card Design Principles

Determining Card Data

The first step in card design requires a study of the final report that is to be printed from the card and a listing of data needed for it. Next the procedure is studied, and any data needed for processing but not for the report is added to the same listing. Every item is recorded on a worksheet. Provision must be made for recording in the card all data that is listed, unless it is calculated or otherwise generated.

A check should be made that the necessary reference data is included. Reference data should be sufficient to:

1. Identify the transaction with the original source document from which it was created.
2. Indicate the date on which the transaction occurred.
3. Establish some reference, such as invoice number, batch number, account number, or salesman number.

Care should be taken to avoid duplicate or unnecessary reference data.

Determining Field Size

The number of positions required to record each type of information should be determined.

The size of the field for card codes, invoice number, and account number is determined by the largest single number to be recorded. With quantity and amount fields, the largest amount that will occur on a reasonably frequent basis must be determined, rather than the largest it could ever be. If the largest possible amount is known and its chances of occurring are rare, multiple cards may be punched for the transaction.

After all card data is listed, the number of columns required should be added. If this is between 80 and 100, it may be possible to reduce it to 80. If it is over 100, an additional card is evidently required. At this point a check should be made to see whether the fields can be rearranged so that all transactions need not have multiple cards, but could have if necessary. Master information can be punched in one card and variable information in the other. Sufficient reference information must be included in the second card if sorting is required.

Some techniques to be considered for reducing the number of card columns are:

- Reduce the size of reference fields by repeating the numbering series more frequently. For example, invoice number may start with 1 each quarter instead of each year.
- Record in the eleventh and twelfth punching positions various codes that may be using a separate punching position.
- Avoid unnecessary data: for example, the use of both an order number and an invoice number may not be necessary if one will provide adequate reference to the other.
- Reduce the size of reference fields by recording. It may be possible to eliminate several positions.
- Reduce the number of columns required for recording reference data by ignoring positions that are not essential for this purpose.

If more than one card is to be used to hold a "record", the division of information between the cards can be made on the following bases:

1. Place constant information in one card (master) and temporary information in the second card (detail).
2. If more than one source document is used, place the information of each document on a separate card and code the cards.
3. When one transaction affects two different accounts, design a card for each account with differing degrees of detail as required by each account.
4. For printing a bill, order, or other notice, design a card for each section of the form. Some of these cards (for example, name and address cards, constant data cards) can be reused.

Determining the Sequence of Fields

Four basic factors are involved in determining field sequence:

1. Sequence of data on the source document from which the new card will be punched
2. Machines and programs used to process the new cards
3. Manual operations in which the new card will be used
4. Location of identical data in other cards with which the new one will be processed

Keeping the sequence of fields similar to the order in which the data is read from the source document will make the keypunching operation faster and more accurate. This is especially important since keypunching is a manual operation and therefore subject to far greater fluctuation in
speed and accuracy than the subsequent mechanized operations. The sequence of fields can be arranged to take maximum advantage of machine characteristics. Specifically, field sequence can be designed to maximize the usage of card punches, sorters, computers (see 20.30.10), control panel wiring, or the manual handling of cards. Placing data in the same columns of the new cards as used in other cards ensures that sorting and controlling the data can be speedily performed when the cards are processed together. It also simplifies control panel wiring where cards are processed by standard punched card machines. If data on the new card is to be checked visually by manually fanning a deck of cards, the fields for that data should be located at either the left or right end of the card.

Using a Card Layout Form

A multiple-card layout form (X24-6599) should be used when planning several cards simultaneously or when planning a new card that will be used with existing cards. The use of this form makes it easy to align those fields that are common to more than one card, where this is desirable. It also makes working with the formats easier, since they are on one sheet of paper and can be compared with one another.

Designing the Card Form

After field size and sequence are established, the design of the card itself can be done. This is usually drawn on a special form considerably larger than the punched card. Photographic reduction is used to create the proper-size print plate (called an "electroplate", or "electro").

It is not always necessary to design a card form for each card used in an application. Where the cards are used only within your data processing department, are interpreted, and are needed only in small quantities, it may be advantageous to use a standard card form, such as the IBM 5081.

Certain principles in the design of card forms should be kept in mind:

1. Field and box headings should be explicit and force writing into desired locations.
2. Adequate space should be allowed for accommodating written information.

3. The right-hand side of a box containing handwritten information should be at least five columns to the left of the columns in which it is to be punched. This is so that the data will not be obscured by the punch station of the card punch machine when it is time to punch it.
4. Information to be punched should not be handwritten along the bottom edge of a card, since the shield on the IBM card punch obscures the lower 1/8" of the card.
5. Field headings should be above the zero row of a card unless interpretation or printing of punches prevents it.
6. Headings and interpreted data should be kept between rows, so that punches will not obliterate them.
7. Preprinted decimals and commas should be placed where dollar amounts will be interpreted.
8. Colored cards, colored stripes, and corner cuts may be used for visible distinction between cards. Also, an identifying punch (called a "key") may be used.
9. Card column numbers should be preprinted where possible and digits should be placed where the numbers can be punched. These aid in reading the punches in a column.
10. Mark-sensing fields should be placed on the right-hand side of a card, so that the card can be easily held and marked.
11. Card or company names should be printed on the ends of a card.
12. When coded punches are used, decoding abbreviations should be preprinted on the card.
13. Where no more than 40 columns are needed, a sectional or "tumble" card may be designed in which the layout in columns 1-40 is duplicated upside-down in columns 41-80. This allows the card to be used twice and cuts card costs in half.

Testing the Card Layout

After the card layout is developed, the fourth and final step in card design is performed — namely, testing the card for its effectiveness. For the test, the new design must be laid out on several cards and the cards must be used in their designated procedure.
DESIGN OF DISK DATA FILES

Introduction

The formats of cards and forms are the tangible types of input/output. You still must design the intangible record formats.

Your 1130 Computing System is concerned with two different intangible records: those in core storage and those on the disk cartridge. Although the storage media are different, the design considerations are the same.

The items discussed in this section concern the components of disk records, the order of the components, and groups of records. Considerations covered include growth, organization, and content.

More detailed information on many of the topics covered here may be found in section 80.
Data

The first step in file design requires a study of all procedures that utilize the file. On the basis of these studies, record each necessary item on a worksheet like the one illustrated in Figure 20.1. Indicate the type of information, the frequency of occurrence, and the sequence in the source document, if applicable. The following should be done:

- Check that the necessary reference data is included, if this is a source file.
- Weigh the effects of media storage costs vs program execution time for constant-type data, such as tax-exempt dollars in payroll.
- Include fields obtained by processing, if the results must be recaptured later.
- Examine all applications that utilize the file, in order to prevent omission of necessary data.
- Explore future requirements of the current procedures. For example, it might be judicious to include an additional deduction field in your payroll application.
- Determine any additional information needed for planned applications. It may be more practical to include an extra field now than to reorganize your files later.
- Study the feasibility of consolidating existing data files into a single data file to eliminate duplication of common information, if such a combined record would not too adversely affect the running time.
- Ascertain whether material needed in a new application, for which the data file is to be designed, is available already in an existing data file.
- Verify that the data file, when set up, will contain all the basic information to meet the requirements of all persons who will be using the products resulting from the file processing.
- Consider file maintenance and audit control.

<table>
<thead>
<tr>
<th>Process Cycle</th>
<th>Record Characteristics</th>
<th>File Dynamics</th>
<th>File Media Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Information Required for Processing and Reporting | Type of Information Required | Field Size | Sequence | REMARKS |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIAL</td>
<td>TRIAL</td>
<td>Trial</td>
<td>FINAL</td>
<td>IN SOURCE DOC</td>
</tr>
</tbody>
</table>

![Figure 20.1. File design worksheet](image)
Field Size

The number of positions required to record each item of information should be determined and entered on a form similar to that shown in Figure 20.1.

Type of Field

Control and indicative data field size should equal the total number of digits in the largest single item to be recorded in the particular field. Occasionally, to conserve storage, the high-order digits may be disregarded for a field such as order number.

Quantitative data field size may equal the total number of digits in the largest amount to be recorded, or the number of digits that will occur with reasonable frequency. Procedures can be developed to handle the rare exceptions.

Recording Medium

Since some media, such as cards and disks, contain a fixed number of positions per unit of storage (card field or disk sector or track, etc.), it is essential to consider this overall limit in order to design efficient and practical records.

Example:

On the 1130, your disk records are automatically "blocked" within 320-word sectors. A 55-word record will be blocked 5 records to the sector with 320-(5x55) or 45 words unused. Rather than waste these 45 words, you might as well increase the size of the record to 64 words, which will still allow 5 per sector (5x64 = 320) with no waste. Or, if possible, reduce the record size to 53 words, which permits 6 records per sector.

File Size (Total Number of Records)

Since the field size affects the total record size, all unnecessary positions should be eliminated to decrease I/O time and storage media requirements.

Future Requirements

If the demands to be placed on the information indicate an impending need for another position, it would be easier to incorporate the additional character in the design phase so as to avoid reprogramming and a patched-up record layout.

Field Compaction Techniques

Because a reduction in the length of a record produces such positive results as an increase in DASD packing and a decrease in time to read and/or write, field compaction techniques should be investigated and the cost of the technique evaluated as each file is designed. Some methods to consider for reducing the number of positions are found in 80.60.00.

A given compaction technique must be evaluated for:
1. Amount of core storage required to hold the encode-decode instructions
2. Encode-decode subroutine timing requirements
3. Compaction percentage achieved
4. Compatibility with programming systems
5. Retention of collating sequence
6. Retention of fixed field length
7. Effect on the overall system, including related clerical functions

Some of these methods are discussed in detail in section 80. For a discussion in depth of compaction techniques see Record Compaction Techniques (E20-8252).
Data Sequence

Data sequence is most critical for those files that work with source documents. Card punching, terminal operations, etc., being manual operations, are subject to the greatest variation in rate of production. Anything that simplifies these functions tends to ensure a faster and more accurate operation. The following are points to bear in mind:

- Recording of data in the same order as that in which it is normally read. If the data sequence is considerably different from that on the source document, it may be necessary to redesign the source document and retrain personnel. If the file is to be used as input to a serial I/O unit, such as disk to card, the sequence is dictated mainly by the sequence desired on the output unit.
- Location of like fields in the same relative record positions in files that work together. This ensures that sorting and controlling can be accomplished if the file is contained in cards; it also facilitates programming.
- Placement of sorting fields adjacent to one another, with the minor code on the right and each progressively higher code to the left. Although sort programs can operate on multiple-control fields, time is used to extract and combine fields into a single key.
- Compatibility with computer characteristics so that data sequence does not affect processing speed.
- Arrangement of alphabetic/alphanumeric data in one area of the record. This facilitates handling of data, particularly in fixed-word-length machines, such as the 1130, and permits minimum core and media requirements.
- Adherence to requirements of programming systems.
File Organization

For strictly card- and/or paper-tape-oriented systems, file organization normally is sequential. Therefore, the following discussion of indexed sequential (as in an encyclopedia) vs random organization (as in shuffled playing cards) is oriented mainly toward the design of disk data files.

Indexed Sequential Advantages

- Both sequential and random transactions can be handled effectively in most cases.
- Reports arranged in data file sequence can be obtained without sorting.
- Control over both the processing and the stored file can be more positive.
- Less storage space is required.
- Frequently the entire file need not be online simultaneously.

Indexed Sequential Disadvantages

- More core storage may be required because of index handling routines.
- Process time is greater for random input because of index file seeking and processing.

Random Advantages

- Less core storage is required normally.
- Process time is less for random input.

Random Disadvantages

- To maintain access requirements, frequent reorganization may be necessary if the file is dynamic.
- Extensive key analysis and development of address conversion routines probably are required for implementation.

A detailed description of these techniques may be found in section 85.
Record Format and Blocking

To select the record format and blocking, each of the following factors must be considered:

1. File boundaries. Cards are limited to 80 columns of punched data, while the disk has 320 words that may be recorded on each sector.

2. Core storage requirements. Since IOCS handles physical records for I/O operations and contains a core storage area large enough to accommodate the physical record, you must supply a core storage area for a logical record. In addition, for efficient operation, multiple I/O areas may be required for the I/O devices.
File Processing

Before the file design is finally determined, the run time and associated costs should be calculated for the entire system. The results must be evaluated to determine whether the original design objectives have been met. If the system is I/O-limited (that is, if I/O time exceeds process time), the following approaches may be considered:

- Create a second master file splitting away from the main master file those fields not required on the primary runs. For example, name and address records could be kept in a separate name and address file. This new file would be used perhaps only as output documents are printed.

- Extract from the master file the active records for processing. This method is useful if the ratio of active master records to total master records is very low.

- Increase the number of input buffers. If the activity rate is low and processing time per hit is high, more process time can be overlapped if the input is queued in additional buffers. If process time requires 250 milliseconds while an input area can be filled in 50 milliseconds, there will be 200 milliseconds of unoverlapped process time per hit, with two input areas. If the number of input areas were increased to four, only 100 milliseconds would not be overlapped.
File Control

The design of a data file cannot be divorced from the environment in which the file must function. Some of the considerations of file control and maintenance are now discussed.

Data Validation

The entry of incorrect data into a file should be prevented. The following techniques may be used to control the accuracy of input data:

1. Precoded forms, or standardized and simplified forms, which reduce the possibility of error at the point of origin of the data.

2. Batch controls that establish totals for a given group of records to detect the loss or distortion of data during intermediate handling. A batch may consist of a fixed number of items or the transactions that occurred in a given period of time. Typical batch totals are record counts, dollar or quantity amounts, and "hash" totals of significant data, such as wage rates. Frequently, batch totals are recorded in a trailer record to provide automatic zero-balance checking.

3. Turnaround documents, such as prepunched remittance forms, which require little or no extra recording and a minimum of handling.

4. Character checking, which determines whether the data in given positions of the record contain permissible characters. This type of check can be used to ensure that the proper algebraic sign is present for the type of transaction or that alphabetic data is not included in numeric fields, and vice versa, or that data is present where required (not blank).

5. Field checks that examine the content of a field for certain characteristics. These include:
   - Limit checks, which determine whether data is within a prescribed range. Such checks can apply to fields such as employee's wage rate, amount of gross pay, etc.
   - Historical checks, which use prior experience as a basis of validation. The public utility industry often compares, for reasonableness, prior consumption for a year or more against current usage.
   - Validity checks, which compare the content of a field against a list of existing "good" numbers. This prevents posting to nonexistent account numbers. Matching by control key against a master file indicates duplicate and missing numbers.
   - Logical relationships, which determine whether the items of input data have a logical relationship to one another or to the file they affect. For example, if an employee adds a bond deduction, a bond denomination is also required.

   - Self-checking numbers, which detect incorrect identification numbers (such as account number, employee number, etc.) by performing certain mathematical calculations on the base number and comparing the resulting digit against a check digit appended to the base number.

Operating Controls

The following controls are common methods used to detect errors caused by poor operator performance, equipment failure, or malfunctioning programs:

1. Disk cartridge ID checking, which verifies that the proper cartridge is online before any processing can take place.

2. Record counts, which check that the numbers of records before and after processing are the same, in order to guard against accidental loss of a record.

3. File totals, which ensure that the file is in balance in light of the transactions just processed. For example, the previous file total for a given field plus the net change represented in the transactions should be equal to the sum of the individual record fields after the transactions are processed.

4. Intervention logging, which records through the console printer any intervention by the operator.

Error Analysis

The file control techniques suggested above indicate the wide variety of methods available. Selection of the specific control procedures depends on such factors as the frequency of possible occurrence, the results if the error were allowed to enter the system, and the chance that the error might remain undetected even through later operations. All errors should be logged indicating the nature and the cause. A review of these error logs can serve as a guide to management to increase or decrease error control.

When errors are detected, any of the following procedures can be used:

1. Programmed halts, where the computer is halted by detection of certain conditions, and the operator follows prescribed steps dependent upon the nature of the halt. The trend is away from programmed halts to eliminate operator intervention.

2. Bypass procedures, where the error condition is recorded on some output medium, such as paper tape or console printer, for later analysis, and the computer continues without stopping.
3. Suspense accounts, where totals are posted for invalid records in order to keep in a single account all items requiring analysis.

Audit Trail

An audit trail may be defined as the means whereby the source transaction and its corresponding supporting documentation can be related to processed data. Although the audit trail may be a by-product of normal processing, it may sometimes be additional. The requirements of the auditor should be discussed to provide the necessary historical information trail.

Reconstruction

If the information on a file is mutilated, the need for reconstruction arises. The method selected depends upon such factors as job priority, the time and cost required to provide reconstruction data, and the time and cost required to perform the reconstruction. Listed below are several approaches:

1. Periodically, a dynamic disk file should be copied (dumped) on paper tape, on cards, or on another disk. Often, the copy can be made as a by-product of a periodic run. All transactions since the last dump must be retained to update the copy to current status.

2. To avoid reprocessing of all transactions since the last dump, write the updated records on paper tape or cards as the transactions are processed against the file. In sequential processing, only one paper tape or card record per active disk record is written. In case of reconstruction, the record with the most recent status can be used to replace the corresponding record on the dumped file.

3. If no output unit is available to record the updated records, as suggested above, the master can be flagged, and on a later run the flag can signal a copy operation for a given record. This technique requires a rewrite to the file for removal of the flag.

4. The contents of a static file should be available either by copying to another disk or by dumping onto paper tape or cards that may be used later to reload the mutilated file.
PAYROLL EXAMPLE

Narrative

Note: All of the pages in the following example represent material that you should have developed by this point in the installation of your system. When completed, the material becomes a part of your system documentation (see section 35).

* * *

The corporation consists of six manufacturing plants, engaged in the fabrication of Liquid Dairy Product Packaging in Ohio, Indiana, West Virginia, and Texas. The payroll system was designed to accommodate all six plants, which have separate bookkeeping records. However, the accounting functions are centralized in one location. Communication is by phone and mail.

The system consists of 16 programs.

The files creation program is first. Data decks are keypunched for each individual, in sets, by plant. The data is edited and, when correct, is loaded on the disk by PAY01. Three files are created: a master file, an index file, and a plant information file. A second data deck with employee clock number and name is loaded onto the master file by PAY02.

Changes to the disk information are made by PAY03. Documents, received from personnel departments at the individual plants, are checked, summarized, keypunched, and verified. Time sheets, submitted by the plant payroll departments, are keypunched and verified. All these cards are processed by PAY16, which edits and generates control totals. PAY04 then processes these cards, performing all payroll calculations. Cards are read, pay is computed, disk files are updated, and cards are extended with current pay figures. After all cards are processed, a payroll register is printed.

Checks are printed by PAY05. A header card is read and the checks are printed from the disk file. PAY06 lists the check register from the disk file. If an error is made in computing pay, PAY11 provides the means of voiding checks. The extended time cards from PAY04 are read in and the affected employee records are reset. The above are weekly runs.

At month end, registers are prepared showing each individual's deductions for the month:
- PAY13 writes union dues register.
- PAY14 writes credit union register.
- PAY15 writes stock deductions register.
- PAY12 resets charity deductions code.

At the end of the quarter and at the end of the year, PAY07 and PAY08 are used to balance the disk files to control totals.
- PAY09 produces the 941 tax report.
- PAY10 produces a tax worksheet used to determine tax liability.
Card Forms and Console Keyboard Input

**PAY01**
- Plant no. - 1 digit - keyboard
- Week no. of month - 1 digit - keyboard
- Check no. - 2 digits - keyboard
- Name - 18 blanks - keyboard
- Plant name - 32 characters maximum - keyboard

Figure 20.3 - card

**PAY02**
- Plant no. - 1 digit - keyboard

Figure 20.4 - card

**PAY03**
- Plant no. - 1 digit - keyboard
- Social Security Number, if changed - keyboard
- Figure 20.5 - card
- Figure 20.6 - card

**PAY04**
- Figure 20.7 - card
- Check no. - 5 digits - keyboard
- Week no. of month - 1 digit - keyboard
- Maximum check amount allowed - 5 digits - keyboard

Figure 20.8 - card

**PAY05**
- Check no. - 5 digits - keyboard
- Check maximum amount - 5 digits - keyboard
- Clock no. (if requested) - 4 digits - keyboard

Figure 20.7 - card

**PAY06**
- Figure 20.7 - card

**PAY07**
- Plant no. - 1 digit - keyboard

**PAY08**
- Figure 20.10 - card
- Figure 20.11 - card
- Figure 20.6 - card

**PAY09**
- Figure 20.12 - card
- Figure 20.13 - card
- Figure 20.14 - card
- Figure 20.15 - card
- Figure 20.16 - card

**PAY10**
- Figure 20.10 - card
- Figure 20.6 - card

**PAY11**
- Figure 20.7 - card
- Figure 20.9 - card
- Figure 20.6 - card

If requested:
- Insurance deduction - 4 digits - keyboard
- Stock deduction - 4 digits - keyboard
- Charity deduction - 4 digits - keyboard
- Miscellaneous deduction - 4 digits - keyboard

**PAY12**
- Plant no. - 1 digit - keyboard

**PAY13**
- Plant no. - 1 digit - keyboard
- Individual amount for a plant - 4 digits - keyboard

**PAY14**
- Plant no. - 1 digit - keyboard

**PAY15**
- Plant no. - 1 digit - keyboard

**PAY16**
- Figure 20.7 - card
- Figure 20.8 - card
Figure 20.4.

![Table and Diagram]

Figure 20.5.
Figure 20.6.

<table>
<thead>
<tr>
<th>Plant No.</th>
<th>Check Date</th>
<th>Earnings Date</th>
<th>Total Clock Numbers</th>
<th>Total Regular Hours</th>
<th>Total Overtime Hours</th>
<th>Total Bonus Hours</th>
<th>Total Special Earnings</th>
<th>Blank</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

Figure 20.7.
<table>
<thead>
<tr>
<th>Clock No.</th>
<th>Regular Hours</th>
<th>Overtime Hours</th>
<th>Bonus Hours</th>
<th>Special Earnings 0</th>
<th>Special Earnings 1</th>
<th>Special Earnings 2</th>
<th>Blank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1 2 3 4 5</td>
<td>6 7 8 9 10</td>
<td>11 12 13</td>
<td>14 15 16 17</td>
<td>18 19 20 21</td>
<td>22 23 24 25</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2 3 4 5 6</td>
<td>7 8 9 10 11</td>
<td>12 13 14</td>
<td>15 16 17 18</td>
<td>19 20 21 22</td>
<td>23 24 25 26</td>
<td></td>
</tr>
</tbody>
</table>

Figure 20.8.
Figure 20, 10.

Figure 20, 11.
Figure 20.16.
Console Printer and Line Printer Forms for Output

PAY01  None
PAY02  None
PAY03  None
PAY04  Figure 20.18
      Figure 20.19
PAY05  Figure 20.19
PAY06  Figure 20.20
PAY07  Figure 20.21
PAY08  Figure 20.17
PAY09  Figure 20.22
PAY10  Figure 20.23
PAY11  Figure 20.18
PAY12  None
PAY13  Figure 20.24
PAY14  Figure 20.25
PAY15  Figure 20.26
PAY16  Figure 20.27
<table>
<thead>
<tr>
<th>LINE DESCRIPTION</th>
<th>FIELD HEADINGS</th>
<th>WORD MARKS</th>
<th>IBM 407, 408, 409, 1403, 1404, 1443, and 2203</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1403 Models 1 &amp; 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 407, 408, 409, and 1403 Models 6 and 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1403 Models 2, 3, 5, N1 and 1404</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1442 Models 1, N4, and 2203</td>
</tr>
</tbody>
</table>

**GLUE**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DESCRIPTION FIELD HEADINGS**

- **FACTOR PAYROLL**: COMPANY NAME IS PRINTED HERE
- **W/xx-xx-xx**: PAGE NO XX
- **NAME**:
- **SPECIAL HOLIDAY VACATION**:
- **SICK**:
- **GROSS**:

**IBM 1403 Models 1 & 4**

- **IBM 407, 408, 409, and 1403 Models 6 and 7**
- **IBM 1403 Models 2, 3, 5, N1 and 1404**
- **IBM 1442 Models 1, N4, and 2203**

**Print Soon**

---

*Figure 20.18. (Cont)*
Figure 20.19

INTERNATIONAL BUSINESS MACHINES CORPORATION

PRINTING SPACING CHART

IBM 407, 408, 409, 1403, 1404, 1443, and 2203

Print Span:

<table>
<thead>
<tr>
<th>GLUE</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IBM 1403 Models 1 & 4

IBM 407, 408, 409, and 1403 Models 6 and 7

IBM 1403 Models 2, 3, 5, 6, and 1404

IBM 1443 Models 1, N1, and 2203

...
### Printer Spacing Chart

#### International Business Machines Corporation

**IBM 407, 408, 409, 1403, 1404, 1443, and 2203**

<table>
<thead>
<tr>
<th>Print Span</th>
<th>IBM 1403 Models 1 &amp; 4</th>
<th>IBM 407, 408, 409, and 1403 Models 6 and 7</th>
<th>IBM 1403 Models 2, 3, 5, N1 and 1404</th>
<th>IBM 1443 Models 1, N1, and 2203</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 20.**

[Grid Image of Printer Spacing Chart]

---

For general printing, this type has been reduced to fit on the page. For complete information, refer to the documentation provided with IBM equipment.
**IBM LINE DESCRIPTION**

**FIELD HEADINGS/WORD MARKS**

<table>
<thead>
<tr>
<th>PRINT SPACING CHARACTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRINTER SPACING CHARACTERS</td>
</tr>
<tr>
<td>IBM 407, 408, 409, 1403, 1404, 1443, and 2203</td>
</tr>
<tr>
<td>Print Span:</td>
</tr>
</tbody>
</table>

**Figure 20.**

- IBM 407, 408, 409, 1403, 1404, 1443, and 2203
- IBM 1403 Models 1 & 4
- IBM 407, 408, 409, and 1403 Models 5 and 7
- IBM 1403 Models 2, 3, 5, 6 and 7
- IBM 1443 Models 1, 10, and 2203

---

**Print Span:**

- IBM 407, 408, 409, 1403, 1404, 1443, and 2203
- IBM 1403 Models 1 & 4
- IBM 407, 408, 409, and 1403 Models 5 and 7
- IBM 1403 Models 2, 3, 5, 6 and 7
- IBM 1443 Models 1, 10, and 2203

---

**Produced for each plant.**
<table>
<thead>
<tr>
<th>LINE DESCRIPTION</th>
<th>FIELD MARKING/WORD MARKS</th>
<th>8 Lines Per Inch</th>
<th>IBM 407, 408, 409, 1403, 1404, 1443, and 2203</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1403 Models 1 &amp; 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 407, 408, 409, and 1403 Models 6 and 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1403 Models 2, 3, 5, NT, and 1404</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1443 Models 1, NT, and 2203</td>
</tr>
</tbody>
</table>

**Figure 20, 23.**
Figure 20. 24.

<table>
<thead>
<tr>
<th>LINE DESCRIPTION</th>
<th>FIELD HEADINGS/WORD MARKS</th>
<th>B Lines Per Inch</th>
<th>IBM 407, 408, 409, 1403, 1404, 1443, and 2203</th>
<th>Print Scan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1403 Models 1 &amp; 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 407, 408, 409, and 1403 Models 6 and 7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1403 Models 3, 5, 6, 7 and 1404</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1403 Models 1, 4, 6, and 7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1403 Models 3, 4, 6, 7 and 1404</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1403 Models 1, 4, 6, and 7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1403 Models 3, 4, 6, 7 and 1404</td>
<td></td>
</tr>
</tbody>
</table>

GL DUE

<table>
<thead>
<tr>
<th>COMPANY NAME IS TO PRINT HERE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
</tbody>
</table>

TOTAL XXXX, XX

Figure 20. 24.
<table>
<thead>
<tr>
<th>LINE DESCRIPTION</th>
<th>FIELD HEADINGS/WORD MARKS</th>
<th>8 Lines Per Inch</th>
<th>IBM 407, 408, 409, 1403, 1404, 1443, and 2203</th>
<th>Print Span</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL NAME</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREDIT UNION DEDUCTIONS</td>
<td>COMPANY NAME IS TO PRINT HERE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>AMOUNT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XXXX</td>
<td>XXXX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>XXXX</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 20, 25.
<table>
<thead>
<tr>
<th>LINE DESCRIPTION</th>
<th>FIELD HEADINGS/WORD MARKS</th>
<th>8 Lines Per Inch</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM 407, 408, 409, 1403, 1404, 1443, and 2203</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**GLUE**

<table>
<thead>
<tr>
<th>Column</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>XXXXX XX XXXXX XX XXXXX XX XXXXX XX XXXXX XX XXXXX XX XXXXX XX XXXXX XX XXXXX XX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 20, 26.**

**Figure 20, 27.**
Disk Record Formats

Employee File - Figure 20.28
Index to Employee File - Figure 20.29
Company Record in the Corporation File - Figure 20.30
EMPLOYEE information record starting at 109 and continuing thru 156 is current information.

Year-to-Date Information

Quarter-to-Date Information

Proceeding Status, Date Code, Gross, PayRate, OT Rate, Regular Hours, Bonus Hours, Regular Earnings, OT Earnings, Bonus Earnings, Other Earnings, Code, Holiday Pay, Vacation Pay, Sick Pay, Net Pay, FICA

For Growth of Record

Figure 20, 28.
Each record is composed of 1 word. The number of records in the file is the number of employees in the plant plus 25%. The last entry is the record number of the last clock number entered.

Figure 20.29.

This is the plant information record.

Figure 20.30.
This page intentionally left blank.
System Flowchart

Employee Earnings Record

Keypunch & Key-Verify Clock No. and Name

Clock No. and Name

Keypunch & Key-Verify All but Name

Totals on Adding Machine

Keypunch & Key-Verify Control Totals

Control Totals

Zero Balance Totals

PAY 16 INPUT EDIT

Balance to Totals & Correct as Necessary

Out of Balance

PAY 02 ADD NAMES

Disk Payroll File

PAY 01 FILE CREATE

Control Totals

File A

Store

Disk Payroll File

Clock No. and Name

File B

File create (initially and as necessary)
File changes (weekly)
Weekly Time Sheets

Totals on Adding Machine

Keypunch & Key-Verify Details

Details

Keypunch & Key-Verify Control Totals

PAY 16 INPUT EDIT

PAY 04 CALCULATION

Balance to Totals; If Incorrect, Go to E

O.K.

Control Totals

Disk Payroll File

Zero Balance Totals

Balance to Totals & Correct as Necessary

Details

Distribute Payroll Register

Disk Payroll File

To B

To C

Control Totals

File D

Payroll calculations and register (weekly)
Print paychecks (weekly)
Only When Totals Do Not Balance

- Destroy Checks
- Disk Payroll File
- Control Totals

PAY 11 VOID CHECKS

- Disk Payroll File
- Control Totals
- Details

File D

Details

Payroll check voiding (as necessary)
Payroll file audit, 941, and tax worksheet (quarterly)
Print W-2 reports (annually)
Individual Payroll Record

PAY 08 INQUIRY

Balance to Totals; If Correct, Go to E

Determine Change Required

Use PAY 16 & PAY 03 to Change the Disk Payroll Record

Does this correct original error? If not, Go to E

Return to Print Where Error Occurred

Select Desired Clock Number Card

Clock Number

Last Week's Payroll Register

Weekly Time Sheets

Section 20 Subsections 50 50 Page 09

Error detection and correction (as necessary)
Remember, all of these pages are developed by this point in your system design. In addition, they now become a part of your system documentation (see Section 35).
LANGUAGES SELECTION

Introduction

Now that your system has been specified, the implementation of the design must be considered. Since you will be writing a program, the logical question is "What language shall I use?"

IBM supplies and supports a wide variety of programming languages and application programs for the 1130 Computing System. Among the programming languages (Type I programs) are:

- 1130 Assembler Language
- 1130 FORTRAN

Some of the application programs (Type II programs) are:
- Continuous System Modeling Program (CSMP)
- Data Presentation System (DPS)
- Linear Programming - Mathematical Optimization Subroutine System (LPMOSS)
- Mechanism Design System - Gears and Springs
- Civil Engineering Coordinate Geometry (COGO)
- Numerical Surface Techniques and Contour Map Plotting
- Programs for Optical System Design (POSD)
- Programs for Petroleum Engineering and Exploration
- Project Control System (PCS)
- Route Accounting for Dairies and Bakeries
- Scientific Subroutine Package (SSP)
- Statistical System
- Structural Engineering Systems Solver (STRESS)
- Type Composition
- Work Measurement Aids
- Commercial Subroutine Package (CSP)

Your IBM representative can help you determine which programming language or application program should be used to implement your system.

In addition to these two types of programs, IBM's Program Information Department maintains a library of contributed programs and distributes these programs to interested parties. These are contributed to the library by:

1. IBM employees (Type III programs)
2. IBM customers (Type IV programs)

Type II and type IV programs have been submitted to the Program Information Department for general distribution in the expectation that they may prove useful to other members of the data processing community. The programs and documentation are, essentially, in the author's original form and have not been subjected to any formal testing. IBM serves only as the distribution agent. It is your responsibility to determine the usefulness and technical accuracy of the programs in your own environment. Unlike programming systems (Type I) and application programs (Type II), these programs are not part of the IBM support package.

The remainder of this section elaborates on each of the programming languages and application programs and discusses some of the considerations in answering "Which do I use?"
Programming Languages

Assembler Language

The IBM 1130 Assembler Language, while similar in structure to machine language, replaces binary instruction codes with symbols and uses labels for other fields of an instruction. Other features, such as pseudo operations, expand the programming facilities of machine language. Thus, the programmer has available, through an assembler language, all the flexibility and versatility of machine language, plus facilities that greatly reduce the machine language programming effort.

The IBM 1130 Assembler Language has two parts: the symbolic language used in writing a program and the assembler program that converts the symbolic language into machine language. An additional component is a library of relocatable I/O, arithmetic, and functional subroutines.

Symbolic language is the notation used by the programmer to write (code) the program. A program written in symbolic language is called a source program. It consists of systematically arranged mnemonic operation codes, special characters, addresses, and data, which symbolically describe the problem to be solved by the computer.

The use of symbolic language:
1. Makes a program independent of actual machine locations, thus allowing programs and routines to be relocated and combined as desired.
2. Allows routines within a program to be written independently and causes no loss of efficiency in the final program.
3. Permits instructions to be added to or deleted from a source program without the user having to reassign storage addresses.

The assembler program (processor), supplied to the user by IBM, operates from paper tape, from punched cards, or under control of the 1130 Disk Monitor Systems. It converts (assembles) a symbolic-language source program into a machine-language (object) program.

The conversion is one for one--that is, the assembler produces one machine-language instruction for each symbolic-language instruction.

The IBM 1130 Assembler is a two-pass program. The processor is loaded into the computer and is followed by the first pass of the source program. During the first pass, source statements are read and a symbol table is generated. During the second pass, the source program is read again and the object program and/or error indications are punched into the first 20 columns of each source card. If paper tape is used, the second pass results in the punching of a new tape that contains both source statements and corresponding object information. If disk is used, this becomes a one-pass procedure, the disk being used for intermediate storage. Both card and tape object programs must be compressed (via a Compressor Program supplied with the assembler) into a relocatable binary deck (or tape) before they can be loaded into core storage for execution.

The output from the second pass is called the list deck (or tape) and can be used to obtain a program listing of source statements and corresponding object statements. Use of disk automatically compresses the object program into relocatable (loadable) form. A program listing is an option if the one-pass disk procedure is used.

A library of I/O, arithmetic, and functional subroutines is available for use with the IBM 1130 Assembler.

The user can incorporate any subroutine into his program by simply writing a statement referring to the subroutine name. The assembler generates the linkage necessary to provide a path to the subroutine and a return path to the user's program. The ability to use subroutines simplifies programming and reduces the time required to write a program.

A description of available subroutines is contained in the IBM 1130 Subroutine Library (C26-5929).

FORTRAN Language

FORTRAN (FORmula TRANslation) is a coding system with a language that closely resembles the language of mathematics. It is a system designed primarily for scientific and engineering computations. Since this system is essentially problem-oriented rather than machine-oriented, it provides scientists and engineers with a method of communication that is more familiar, easier to learn, and easier to use than actual computer language.

The IBM 1130 Basic FORTRAN IV Programming System consists of two parts: the language and the compiler. The language is a set of statements, composed of expressions and operators, that are used in writing the source program. The 1130 FORTRAN compiler, provided by IBM, is a program that translates the source program statements into a form suitable for execution on the IBM 1130 Computing System. The translated statements are known as the object program. The compiler detects
certain errors in the source program and writes appropriate messages on the console printer, 1132 Printer, or 1403 Printer. At the user's option, the compiler also produces a listing of the source program and storage allocation.
Application Programs

Continuous System Modeling Program

This program provides engineers and scientists with a simple but versatile tool for solving dynamic system simulation problems. For many problems, this program obviates the need to use an analog computer facility.

CSMP is a "digital analog simulator" program using a block-oriented input language in which the functional blocks represent the elements and organization of an analog computer. A total of 25 standard functional blocks plus the ability to define special functions are provided. The continuous system model may be developed and tested, and results observed in an online interactive mode by means of the console keyboard and output devices. The simplicity of the language statements enables a user to rapidly gain proficiency with the program and facilitates modification of the model via the console. In addition, via the console printer, the beginner is provided instructional comments that can be suppressed as experience is gained. Simplicity and flexibility are the foremost characteristics of the program.

Data Presentation System

This program can present a large variety of data in plotted forms such as graphs, charts, schematics, and modified drawings. It supplies high-quality, hard-copy, graphic output at exceptionally low cost. The system can be used independently as a Graphic Report Generator, or the user can choose one or two levels of subroutines from the system for inclusion in his own graphic output programs. These three levels of access are made even more flexible by several system modification and expansion features. The scope and flexibility of DPS make it valuable in almost every application of the IBM 1130 Computing System.

Linear Programming -- Mathematical Optimization Subroutine System

LP-MOSS provides the 1130 disk user with a simple, efficient means of solving linear programming problems and a means for implementing a variety of mathematical optimization applications.

Mathematical optimization is any mathematical technique for determining the optimum use of various resources such as capital, raw materials, manpower, and plant or other facilities. The technique seeks to attain a particular objective (for example, minimum costs or maximum profit) when there are alternate uses for the resources. Linear programming is the most widely used of these techniques, and has been used to allocate, assign, schedule, select, or evaluate the uses of limited resources for various jobs, such as blending, mixing, bidding, cutting, trimming, pricing, purchasing, planning, and the transportation and distribution of raw materials and finished products.

Mechanism Design System -- Gears and Springs

This program provides design and analysis for five distinct mechanical components used in a wide variety of machines in all industries. Spur and helical gears, compression, extension, and torsion springs are the components covered. The program provides the mechanical engineer and mechanism designer with a low-cost, flexible, easy-to-use program set which will design new parts or analyze existing parts.

The engineer is expected to furnish the problem description in terms of design restrictions and material parameters. This description is in a flexible problem language format which greatly simplifies man-machine communication. Operation can be either by a batch card input mode or in a conversational typewriter input mode. In the latter case, an engineer can readily evaluate parametric changes and truly use the computer as a design tool.

Civil Engineering Coordinate Geometry

COGO is a simple, efficient tool designed especially to assist the civil engineer with a wide variety of geometric calculations. With COGO, the engineer can state his problems using familiar terminology common to the engineering field. No knowledge of traditional programming is necessary.

The civil engineer requires a simple but efficient means to solve geometric problems now being done laboriously by hand. 1130 COGO provides the solution to his problem by allowing the engineer to (1) enter the data for the job into the computer by typewriter or punched cards, using a language with which he is familiar, and (2) to have solutions automatically printed out. COGO is especially useful because it provides the facility for the engineer to try many different methods of solving a problem.

COGO can be used for many different types of jobs, e.g., control surveys, highway design, right-of-way
surveys, bridge geometry, subdivision calculations, land surveying, construction layout.

COGO can, in fact, be used wherever geometric calculation is required.

Numerical Surface Techniques and Contour Map Plotting

This program provides a variety of techniques for describing and operating on surfaces. Surfaces may be described analytically by equations or numerically by sets of data points. In addition, various arithmetic and logical operations may be performed on these surfaces. These techniques may be carried out individually or in various combinations by storing intermediate data in the online disk storage. Final output is commonly in the form of maps drawn by the 1627 Plotter, but may optionally be in card form.

Optical System Design

POSD provides the optical designer with a convenient, efficient design tool. It is in the Computer Aided Design (CAD) category of programs, thus exhibiting a close man-machine relationship throughout the design task. The 1130 Computing System is ideal for this interaction, because it is fast, convenient, and inexpensive to use.

POSD removes the drudgery and error-proneness from the innumerable calculations required in the optical design and evaluation process and allows the designer to spend his time exercising creative and critical judgments. The program gives the designer step-by-step assistance from the very early stages of the design through to the final optimization process. In addition, the designer may evaluate the quality of his design at any time he chooses through many data plot or printout routines or both. Using this program, the optical designer can tackle virtually any lens system, including those requiring a high degree of sophistication, with the assurance that the lens performance will meet specifications in modeling and manufacture.

Programs for Petroleum Engineering and Exploration

Economic Evaluation of Petroleum Projects Program can be used to screen drilling proposals and rank them according to their profitability. Given the investment schedule and production forecast for an exploration and drilling prospect, the programs compute the payout period and rate of return using the discounted cash flow method.

Casing Design Program allows the user to design the most economical combination casing string, in terms of grade and weight, that will meet the requirements of a given well.

Decline Curve Analysis Program computes the coefficients in the equation best fitting past production data and the reserves associated with these data.

Tanner Material Balance Program is an aid in predicting the performance of a reservoir.

Schlilthuis Material Balance Program for a reservoir that is subject to water influx, is evaluated at each past production data point (for up to 28 points). These values are weighted according to oil production and subjected to a least-squares solution to compute a most probable value of the original oil in place.

Two-Dimensional Waterflooding Program allows the user to determine the pressure distribution throughout a reservoir, taking into consideration the effect of water injection.

Gas Deliverability Program allows the user to project the annual rate at which volumes of gas reserves may be received into gathering systems.

Multi-State Flash Calculation Program is a general purpose flash calculation program that can be used for a variety of the computations made by the petroleum engineer. The program may be used to design surface separators or to determine the physical properties of the oil and gas from a surface facility. A laboratory differential liberation may be simulated.

Velocity Functions from Time-Depth Data Program permits a geophysicist to derive a velocity function and to prepare a tabulated time-depth chart from well velocity data.

Wave-Front Ray-Path Determination Program provides a flexible method to compute and tabulate a seismic wave-front ray-path chart; the geophysicist uses such a chart to restore seismic reflections to their true subsurface position.

Synthetic Seismogram Program computes and plots a one-dimensional seismic model from well log data.

Gravity and Magnetics Continuations, Derivatives, and Residual Program provides a method for computing (1) upward and downward continuations of gravity and magnetic fields, (2) first and second derivatives of these fields, (3) residuals of arbitrary type for gravity and magnetic values.
Theoretical Gravity of a 3-D Mass Program allows the user to establish a synthetic gravity anomaly by computing the theoretical gravity of an assumed mass.

Quantitative Log Analysis Program permits the user to compute the porosity and water saturation on prospective hydrocarbon zones in a well, using data from several log combinations.

Dipmeter Program is designed to assist in the analysis of the continuous dipmeter log by calculating the true dip of intervals in a well.

Project Control System

This program provides a basic tool needed by management to fulfill its responsibilities in the planning, supervising, and controlling of project-oriented work. In addition to critical path analysis, the system provides the capability for summarizing externally prepared resource and cost information.

For critical path networks, the 1130 PCS will process 2,000 activities either in the form of precedence lists or in \( ij/PERT/CPM \) notation. Its design allows for a simple approach to networking, but also offers many of the features normally found only in programs designed for large computers.

Route Accounting for Dairies and Bakeries

This is a set of programs offering the functions of route settlement and associated report preparation as required in the dairy and baking industry. Output includes order listings, production requirements, load listings, product load strips, route settlement, and statistical reports.

Scientific Subroutine Package

SSP is a collection of FORTRAN subroutines that provide a major addition to those built into FORTRAN. They are input/output-free, computational building blocks that can be combined with a user's input, output, or computational routines to meet his individual needs. The package has widespread application to the solution of problems in research, development, and design, in both science and engineering, wherever FORTRAN is used.

Statistical System

This is a collection of four major tools: stepwise regression analysis, factor analysis, analysis of variance, and orthogonal polynomial curve fitting. This flexible system accepts user-supplied control cards (and data) that instruct the system to perform one or more of the above analyses.

Structural Engineering Systems Solver

STRESS is a powerful tool for solving structural engineering problems. It is a problem-oriented language that enables the engineer to communicate with the computer even though he has had no previous programming experience.

This program covers many application areas in the field of structural analysis. Most buildings and bridges are designed by consulting engineers or government agencies, but many other types of structures in other industries can also be designed using 1130 STRESS. Some of the other industries and typical applications for each are:

<table>
<thead>
<tr>
<th>Industry</th>
<th>Typical Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerospace</td>
<td>Wing members</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>Conveyor framing, plant design</td>
</tr>
<tr>
<td>Process</td>
<td>Supporting towers</td>
</tr>
<tr>
<td>Utilities</td>
<td>Transmission towers, culvert sections</td>
</tr>
<tr>
<td>Federal</td>
<td>Dam design, ship design</td>
</tr>
</tbody>
</table>

Type Composition

This program extends the speed and flexibility of a digital computer into the composing rooms of the printing industry. Type compositors can use this program to provide significant time savings in transcribing textual material into a form required by linocasting machines for setting type.

The program is designed to allow computer acceptance of perforated paper tape containing (1) the copy that is to appear in print and (2) instructions pertaining to a desired printing format. From the paper tape, a tape suitable for controlling the operations of a linocasting machine is produced and allocated to the proper point in the composing room. The output tape contains the original copy in the form of properly justified lines arranged according to the stylistic and graphic requirements described by the user with the format instructions. The programs are capable of producing justified lines in any format within the inherent limitations of the linocasting machine.
Work Measurement Aids

This program aids manufacturers who need to know the time it should take to manufacture a product. This task, often referred to as work measurement, has traditionally been very time-consuming and expensive. Work Measurement Aids provides two programs to assist in setting time standards. This information also forms the foundation for labor standards, cost estimates, machine operation instructions, and scheduling input. The two programs are:

Machinability, which determines optimum machine tool parameters such as speed, feed, horsepower, tool life, and process time for machining operations.

Work Measurement Sampling, which determines job standards for long cycle operations (over 15 minutes) and the distribution of time to job activities (conventional work sampling).

Commercial Subroutine Package

This program provides the scientific and engineering user with added capabilities for handling functions and techniques common to commercial programming. It is a set of 28 subroutines callable by the FORTRAN programmer in a similar manner to such standard functions as sine, cosine, square root, etc. The subroutines enable the 1130 user to add commercial applications such as payroll, cost accounting, and many others.

The additional functions supplied are variable length alphameric move, variable length alphameric compare, variable length alphameric edit, variable length conversion from EBCDIC to floating-point, variable length conversion from floating-point to EBCDIC, zone manipulation, fill an area with a specified character, stacker select, variable length decimal add, variable length decimal subtract, variable length decimal multiply, variable length decimal divide, variable length decimal compare, sign manipulation, overlapping printing and carriage control, overlapped reading of cards with conversion of card codes, overlapped printing on the console printer, and conversion from one character per word to two characters per word.
Which Programming Language or Application Program Should You Use?

In terms of coding ease and elapsed time from problem definition to operating program, the programming techniques available to you will generally rank as follows:

1. Application programs (except Commercial Subroutine Package and Scientific Subroutine Package)
2. FORTRAN, Commercial Subroutine Package, and Scientific Subroutine Package
3. Assembler Language

The Assembler Language is rarely used, because FORTRAN, augmented by the Commercial Subroutine Package and Scientific Subroutine Package, is more than capable of handling almost all applications, is easier to code, and produces efficient programs.

The brief descriptions given earlier will help you to select the best language in which to program your applications. A preview of the payroll programs given in Sections 25 and 35 will give you a clearer picture of the kind and amount of writing required to code some typical commercial jobs.

In addition, Section 70 discusses FORTRAN, the Commercial Subroutine Package, and how to use these two tools in implementing your system design.
<table>
<thead>
<tr>
<th>Section</th>
<th>Subsections</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

**Section 25: PROGRAM DEVELOPMENT**

**CONTENTS**

<table>
<thead>
<tr>
<th>Content</th>
<th>Subsections</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>25.01.00</td>
<td></td>
</tr>
<tr>
<td>Programming and Documentation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standards</td>
<td>25.10.00</td>
<td></td>
</tr>
<tr>
<td>Program Change Authorization</td>
<td>25.20.00</td>
<td></td>
</tr>
<tr>
<td>Programming Aids</td>
<td>25.30.00</td>
<td></td>
</tr>
<tr>
<td>Documenting Variable Usage</td>
<td>25.30.10</td>
<td></td>
</tr>
<tr>
<td>Modular Programming</td>
<td>25.30.20</td>
<td></td>
</tr>
<tr>
<td>Programming Examples</td>
<td>25.40.00</td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>25.40.01</td>
<td></td>
</tr>
<tr>
<td>Example 1: File Creation</td>
<td>25.40.10</td>
<td></td>
</tr>
<tr>
<td>Example 2: Add Name to the File</td>
<td>25.40.20</td>
<td></td>
</tr>
<tr>
<td>Example 3: Changes to the File</td>
<td>25.40.30</td>
<td></td>
</tr>
<tr>
<td>Example 4: Calculations and</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payroll Register</td>
<td>25.40.40</td>
<td></td>
</tr>
<tr>
<td>Example 5: Check Writing</td>
<td>25.40.50</td>
<td></td>
</tr>
<tr>
<td>Example 6: Check Register</td>
<td>25.40.60</td>
<td></td>
</tr>
<tr>
<td>Example 7: 941 Report</td>
<td>25.40.70</td>
<td></td>
</tr>
</tbody>
</table>
INTRODUCTION

This section is a workbook for the programmer. Primarily by example, and to some extent by narrative, he is furnished with a guide to coding.

First, suggestions are made for the adoption of certain standard practices that will make the programming job easier and the results more uniform. Then follows a series of programming aids.

The bulk of this section is occupied by the final part, a group of examples of coding required to implement a significant part of the payroll system discussed earlier. They will prove useful in providing a starting point for the programmer and illustrating proven programming techniques, rather than in being usable without change for any given installation's system. Note that programs are written at this point in the installation of your system. Also, Variable Summary Sheets are filled in and flowcharts are drawn. These last two items now become a part of your documentation (note references to Section 35).
PROGRAMMING AND DOCUMENTATION
STANDARDS

For a discussion of the documentation that you should have upon completion of a program, see Section 35.

It is advisable to decide on and write down, perhaps following this page, your own standard procedures for handling the situations below. You should have some knowledge of programming before attempting to do this.

1. Alternative methods of handling standard types of errors (for example, missing date card):
   a. Assign a standard halt number.
   b. Assign a standard halt number and error message.
   c. Assign a standard error stacker; do not halt.

2. Standard error messages:
   a. Establish a log of error messages and halt numbers and their meaning.
   b. Standardize spacing, skipping, location, and whether to halt for each standard error message.

3. Standard FORTRAN labels:
   a. Assign a standard symbolic name for each I/O device.
   b. Assign standard field names for fields used frequently.
   c. Assign standard subroutine names for routines used frequently.

4. Record layout conventions:
   a. Define standard heading (for example, date to left, title in center, report number and page number to right).
   b. Define spacing (for example, when listing, single-space detail, double after minor, triple after intermediate and up; when tabbing, single-space after minor, double after intermediate, triple after major and up).
   c. Define how totals are to be indicated (with asterisks or message).
   d. Define how final totals and control totals are to be printed (for example, at bottom of page, on next page).

5. Specify when flowcharts are required for program logic:
   a. When a significant number of GO TO or IF statements are used.
   b. When a complex table lookup is performed.
   c. Whenever the logic of the computation is so complex that another person would have difficulty following it without the aid of a chart (decision tables may be best).

6. Describe how program changes are to be made:
   a. Require changes to be authorized.
   b. Assign all changes to a programmer through the manager of data processing.
   c. Keep track of time spent making program changes by application and by initiator of change.
   d. Require that all necessary documentation be brought up to date.

7. Outline methods of testing programs:
   a. Define conditions in which a test deck is sufficient.
   b. Define conditions in which a program must be production-tested before installation.

8. Standardize writing of specifications:
   a. Establish a standard identification (see the accompanying FORTRAN coding form).
   b. Use a standard form of program identification, such as a three-character application code followed by a two-digit program number (for instance, PAY10, PAY20, BIL10).
PROGRAM CHANGE AUTHORIZATION

All changes to an operating program should be controlled in order to avoid confusion and unauthorized changes. The following sheet is suggested as a means of maintaining control.

<table>
<thead>
<tr>
<th>Application</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requested by:</td>
<td>Date / /</td>
</tr>
<tr>
<td>Description of the Change</td>
<td></td>
</tr>
</tbody>
</table>

Change Authorized by ____________ Date Authorized / / /  
Date Change to be Effective / /  Actual Effective Date. / /  

Programmer: 

Original________________________ Assigned for Change________________________

Date Assigned / /  Date Completed / /  

Systems Design Hours Required

Coding/Debugging Hours Required
### FORTRAN CODING FORM

<table>
<thead>
<tr>
<th>Statement</th>
<th>FORTRAN STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB NAME</td>
<td></td>
</tr>
<tr>
<td>JOB NUMBER</td>
<td></td>
</tr>
<tr>
<td>PROGRAMMER</td>
<td></td>
</tr>
<tr>
<td>DATE CODED</td>
<td></td>
</tr>
<tr>
<td>DATE UPDATED</td>
<td></td>
</tr>
<tr>
<td>FILE</td>
<td>FILE RECORD NO. OF RECORDS</td>
</tr>
<tr>
<td>INPUT FILES</td>
<td>1, 2...</td>
</tr>
<tr>
<td>OUTPUT FILES</td>
<td>1, 2...</td>
</tr>
</tbody>
</table>

* A standard card form, IBM electro 888157, is available for punching source statements from this form.
PROGRAMMING AIDS

Documenting Variable Usage

Especially when writing a large program in FORTRAN, it is difficult to remember the functions for which the variables have been used. This problem may arise during testing, particularly when several programs are being tested at one time. Again, program revision at a later date can be difficult, and the problem is intensified if the revision is being done by someone other than the original programmer.

The Variable Summary Sheet is a suggested form for recording the usage of variables. The sample shown here is related to PAY01 shown later in this section. Both the variables used and the type (I, R, D, A) are indicated in the columns to the left of the form.
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT</th>
<th>OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
</tr>
</thead>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE*</th>
<th>No. of Words</th>
<th>INPUT/TEMP OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKMAX</td>
<td>R I T</td>
<td>10000</td>
<td></td>
<td>0.00</td>
<td></td>
<td>Maximum check amount for a file.</td>
</tr>
<tr>
<td>COMP</td>
<td>R I O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Company name.</td>
</tr>
<tr>
<td>FIBRE</td>
<td>R 0</td>
<td>0.00 0.00</td>
<td></td>
<td></td>
<td></td>
<td>Trade association reports.</td>
</tr>
<tr>
<td>I</td>
<td>I I T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Used in DO loop</td>
</tr>
<tr>
<td>IC</td>
<td>I I N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>ICHCK</td>
<td>I I T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Beginning check number when writing checks.</td>
</tr>
<tr>
<td>ICOL</td>
<td>I I T</td>
<td>250 1</td>
<td></td>
<td></td>
<td></td>
<td>Record number in Employee File. Set up by plant.</td>
</tr>
<tr>
<td>IND</td>
<td>I I T</td>
<td>106 101</td>
<td></td>
<td></td>
<td></td>
<td>File number of index for a plant. P# + 100</td>
</tr>
<tr>
<td>INDEX</td>
<td>I I T</td>
<td>xxxx 1000</td>
<td></td>
<td></td>
<td></td>
<td>Index to plant now being processed</td>
</tr>
<tr>
<td>INIT</td>
<td>I I O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Union initiation fee</td>
</tr>
<tr>
<td>IN1</td>
<td>I I T</td>
<td>250 1</td>
<td></td>
<td></td>
<td></td>
<td>Record number in Indexes to Employee File</td>
</tr>
<tr>
<td>IN2</td>
<td>I I N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN3</td>
<td>I I N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN4</td>
<td>I I N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN5</td>
<td>I I N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN6</td>
<td>I I N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IPD</td>
<td>I I O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Indicates status of record in processing cycle</td>
</tr>
<tr>
<td>ISUPP</td>
<td>I I O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Supplemental sick pay</td>
</tr>
<tr>
<td>ITOT</td>
<td>I I T</td>
<td>1723 0</td>
<td></td>
<td></td>
<td></td>
<td>Account number for posting to General Ledger</td>
</tr>
<tr>
<td>IWEK</td>
<td>I I T</td>
<td>5 1</td>
<td></td>
<td></td>
<td></td>
<td>Week of the month</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
Modular Programming

General

Modular programming is used to divide your problem solution into its logical parts or routines so that each routine may be programmed independently. It enables your complex problems to be divided into many simple sections. A building block program is thereby created that is controlled by a single routine commonly known as the "main line".

A modular program utilizes the same communication system as established by an organization chart. Work assignment decisions are made by the main line routine, which is not concerned with the functions of the processing routines. If for some reason a routine is revised or eliminated, other processing routines within the program are not affected. However, a segment of the main line might be changed.

There are three primary design criteria of modular programming: ease of understanding, ease of program modification, standardization of program construction.

To prepare and use an operational program effectively and efficiently, you must be able to understand the content of the program readily. Ease of understanding is provided in the following three ways:

1. Modular flowcharts. A modular system flowchart gives an overall picture of the major components and structure of the routine; program flowcharts then progress to any desired level of detail, depending on the complexity of the routine. The program coding is referenced throughout.

2. Detailed narrative of each routine. The narrative of each routine states the purpose of the routine, describes the data processed by the routine, and explains each step of the program logic as portrayed by the modular flowchart of the routine.

3. Programming conventions. The use of standard labeling conventions and standard program documentation techniques enables a person unfamiliar with the program to readily understand the program content.

Years of experience have shown that, with 98% assurance, all of your operational programs will require modification and change during their useful life. Ease of program modification is of cardinal importance when your program must be converted to fit a specific new situation. This may be because of changing company policy, varied environmental parameters or different management objectives. Your programmer, then, has the problem of creating a program that can be adjusted to each specific situation. There are two ways of handling this problem.

One is to try to anticipate every type of special situation that might be encountered and write a set of routines to handle each situation. This would require a fantastic ability to forecast the future and would lead to slow, cumbersome programs.

The other alternative is to create a program that can be quickly understood and easily modified to reflect changing conditions. Modular programming aims to accomplish the latter alternative.

Once again, you may more readily prepare and more quickly implement an operational program if all the runs (programs) within your application adhere to a standardized construction. As indicated above, the logical structure of your program must be such that modifications and additions can be easily made.

Consider the problem of multiple routines - for instance, three economic order quantity routines. The normal method of lumping these three routines into a program necessitates setting switches to tell the program which routine to execute at a given time. Any attempt to modify one of the existing routines necessitates trying to extract the routine, patching up the holes in the flow of the program created by the changes, and then fitting the modified routine back in. Anyone who has ever tried to modify a program written by someone else knows how difficult it is to dissect and patch another person's logic if the routines are intertwined.

Using modular programming, each routine is a separate entity. Your main line routine provides the master control that ties all of your individual processing routines together and coordinates their activity.

Modification of routines is simplified. Furthermore, new routines may be added by simply expanding the main line routine to transfer control to the new routine in the proper sequence.

Modular Programming Conventions

Modularity is accomplished by employing the following conventions:

1. The main line
   a. The main line routine makes all decisions governing the flow of data to the proper processing routines.
   b. No processing routine can direct data flow to another processing routine.
c. Input and output functions that are common to more than one processing routine are controlled by the main line routine.

2. Processing routines
a. A separate processing routine is created for each logical segment of the program. It should accomplish one task in its entirety.

b. Each processing routine is complete within itself, with its own defined areas, when such areas are for the exclusive use of that routine. No decision made outside the segment should determine the processing within a segment, and likewise, no decision within a segment should determine the processing outside the segment.

c. Each routine is designed so that it is, in effect, an out-of-line subroutine. Control is transferred to the processing routine from the main line routine, and when the routine has performed its function, it sends control back to the main line routine. Entrance to and exit from the routine never depends on a particular preceding or trailing segment.

d. A processing routine may transfer control to a multiple-use subroutine. When that routine has performed its function, it transfers control back to the processing routine.

e. Input or output functions that affect only one processing routine may be performed by that routine. All segments should contain their own initialization to ensure noninterference with other segments.

f. A debugging aid that is sometimes useful is the inclusion of pauses at the exit of processing routines. During testing, the pause indicates that a particular processing routine has been executed. After the routine is checked out, the pause is removed. The insertion of GO TOs into the program at strategic points may also be used to bypass the testing of particular routines. Action to be taken regarding such PAUSEs and GO TOs must be known and documented before the testing session. This technique tends to make good use of test time.

3. Multiple-use subroutines
a. If the same sequence of statements is used by two or more processing routines, these statements should be established as a multiple-use out-of-line subroutine.

b. A multiple-use subroutine must be well documented for the purpose of program modification. Comments cards should be used to indicate which processing routines call upon each multiple-use routine and to document the linkage established.

Designing a Run

To design a modular program, determine the program variables. List the requirements, elements, and functions of the program as they come to mind, giving no attention to logical order.

Once the variables have been set down, reviewed, and revised, determine the logical order of the processing routines, and design the main line of your program. Construct your main line so that the largest volume of data is processed by the lowest number of instructions - that is, in the fastest possible way. A speedy main line contributes greatly to the throughput capabilities of your program.

Once you have established the logic of your main line, draw the overall, big-picture, system flowchart. Give careful attention to this diagram because it will tend to reveal most errors in logic.

The following components are generally found to be present in the main line of typical programs:

1. Beginning of item. Before obtaining a record, it is often necessary to initialize certain switches, counters, and areas. Generally, fewer instructions are required to initialize before entering a routine than after exiting from it, since routines commonly have several exits.

2. Obtain the item. This segment of the run retrieves the record, sequence-checks the file, and updates the input control.

3. Process the item. The processing of the record is accomplished. The main line transfers control to the proper processing routines in the proper sequence.

4. End of item. Generally, there are a few instructions to be executed just before disposing of a record. The instructions associated with the cleanup work for the present record should not be confused with initialization for the next record.

5. Dispose of the item. This segment of your run generally puts the record in an output file, updates the output controls, and transfers the program to the beginning-of-item routine to start the loop again.

Use the modular technique with a block wherever it simplifies the logic of the processing routine. Each routine should be as efficient as possible.
Look for opportunities to consolidate several in-line routines into one multiple-use subroutine. While sophisticated programming techniques are acceptable, the particular degree of skill and knowledge available to maintain and modify the program should be kept in mind.

The following suggestions may help when programming and documenting:

1. List the functions of your routine.
2. Plan the logic of your routine and sketch a flowchart.
3. Program your routine.
4. Draw the final modular flowcharts of your routine, shown to the necessary levels of detail.
5. Create the test data so that every leg of your routine will be thoroughly tested.
6. Write the detailed narrative of your routine.

It is easier to document your routine when the information is fresh in your mind; furthermore, the documentation thus produced is more meaningful and more comprehensive.

Summary

It has been found that programs employing the modular technique are efficient from the standpoint of both core storage utilization and program execution time. Section 90 illustrates the importance of these techniques.

Furthermore, extremely comprehensive and detailed applications, designed and documented with the use of modular techniques, may be readily understood by non-program-oriented personnel, ranging from company executives to novice programmers.
PROGRAMMING EXAMPLES

Introduction

The examples in this section show various basic programs in the payroll system. Note that these examples are programming illustrations and therefore may not be considered as complete, usable programs. The programs are arranged in the order of their complexity, progressing from the simplest to a complex file-update run with exception reporting.
Example 1: File Creation

This program reads cards containing employee earnings information. The information is edited for reasonableness and then written onto the disk.

The program illustrates a simple single-file at a time run, with a minimum of calculations. The following programming techniques have been used:

1. Documenting with comments. Comment cards have been used to document the program logic. The program name and other indicative information are documented at the beginning of the program. Comment cards describing the processing to be performed are placed before each logical section of the program.

2. One-at-a-time input from the console keyboard. Data items to be read from the console keyboard are requested one at a time (statements 69+1, 69+2, and 69+3). This technique will reduce console input errors and will notify the operator when a requested field has been completed (the keyboard request light will go out).

3. Entering a partial record. Since the complete employee record requires more than 80 card columns, it cannot be punched in one card. The name, which requires 18 card columns, is punched on a second card. However, to prevent a name card and its associated employee record card from becoming separated, the employee name is stored on the disk by PAY02.

4. Editing for reasonableness. Fields on a card which have limits, or a range of values, are checked to ensure that they fall within the range (statements 100 through 109). This provides an effective control of the information being stored on the disk.

5. Program identification numbering. The program identification for the File Creation Program in the Payroll System is PAY01. This method of identification uses a three-character alphabetic abbreviation of the application, followed by the two-digit run number in the application. Identifying programs and documentation in this manner facilitates an efficient system of organizing and filing the documentation and the various decks pertaining to each computer run.

6. Using packed data. To take full advantage of the disk storage available, as much information as possible is packed. This includes the employee and plant name fields. In addition, where possible, some values are compressed by storing them as integers rather than real numbers.

7. Setting up for future reference to the file. The file organization scheme to be used in the payroll system is indexed sequential. This program must create the index, in addition to creating the file. Notice that there is an index entry for each employee. Later programs will be able to locate any employee by simply searching the index in core storage and then reading the employee record. The relative position of the employee number in the index is the record number of the employee in the file.

8. Variable Summary Sheets. These very important forms are present in the following pages. They have been prepared for this program and all other programs in the system.
Start

Initialize Variables

Read Plant No. Week No. Check No.

Setup Name Field

Retrieve Company Name

Read All Information for One Employee

Last Card?

Yes

Initialize Trade Association Information

No

Setup Quarter-to-Date Information

Write to Disk the Index of Employees

Check the Data for Reasonableness

Write the Record to Disk for This Plant

Write Disk Record of All Information for Employee

Stop
<table>
<thead>
<tr>
<th>Entry</th>
<th>Clock No.</th>
<th>Pay Rate</th>
<th>Soc. Security No.</th>
<th>Local Ten</th>
<th>FICA YTD</th>
<th>FIT YTD</th>
<th>Credit Union Deduction</th>
<th>Union Dues</th>
<th>Blank</th>
<th>Blank</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
| 1     | 2 3 4 5   | 6 7 8    | 9 10 11 12       | 13 14 15 16 | 17 18 19 20 | 21 22 23 24 | 25 26 27 28 | 29 30 31 32 | 33 34 35 36 | 37 38 39 40 | 41 42 43 44 | 45 46 47 48 | 49 50 51 52 | 53 54 55 56 | 57 58 59 60 | 61 62 63 64 | 65 66 67 68 | 69 70 71 72 | 73 74 75 76 | 77 78 79 80 | 81 82 83 84 | 85 86 87 88 | 89 90 91 92 | 93 94 95 96 | 97 98 99 100 |}

Note: The table contains placeholders for data entries, indicating where information should be filled in. The placeholders are represented by `0`, `1`, `11`, `12`, etc., to signify the positions where actual data should be entered.
### VARIABLES

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKMAX</td>
<td>R $\frac{1}{3}$ T</td>
<td>100000</td>
<td>0.00</td>
<td>Maximum check amount for a file</td>
<td></td>
</tr>
<tr>
<td>COMP</td>
<td>I 1 G 10</td>
<td>-</td>
<td>-</td>
<td>Company name</td>
<td></td>
</tr>
<tr>
<td>FIBRE</td>
<td>R $\frac{1}{24}$ O</td>
<td>0.000000</td>
<td>0.00</td>
<td>Trade association reports.</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>I 1 T</td>
<td>-</td>
<td>-</td>
<td>Used in DO loop</td>
<td></td>
</tr>
<tr>
<td>IC</td>
<td>I 1 N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
<td></td>
</tr>
<tr>
<td>ICHCK</td>
<td>I 1 T</td>
<td>Set</td>
<td>each run</td>
<td>Beginning check number when writing checks</td>
<td></td>
</tr>
<tr>
<td>ICOL</td>
<td>I 1 T</td>
<td>250</td>
<td>1</td>
<td>Record number in Employee Files, set up by plant</td>
<td></td>
</tr>
<tr>
<td>IND</td>
<td>I 1 T</td>
<td>106</td>
<td>101</td>
<td>File number of index for a plant, Plant + 100</td>
<td></td>
</tr>
<tr>
<td>INDEX</td>
<td>I 250</td>
<td>xxa</td>
<td>1000</td>
<td>Index to plant now being processed</td>
<td></td>
</tr>
<tr>
<td>INIT</td>
<td>I 1 O</td>
<td>0</td>
<td>0</td>
<td>Union initiation fee</td>
<td></td>
</tr>
<tr>
<td>IN1</td>
<td>I 1 T</td>
<td>250</td>
<td>1</td>
<td>Record number in Indexes to Employee File</td>
<td></td>
</tr>
<tr>
<td>IN2</td>
<td>I 1 N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
<td></td>
</tr>
<tr>
<td>IN3</td>
<td>I 1 N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
<td></td>
</tr>
<tr>
<td>IN4</td>
<td>I 1 N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
<td></td>
</tr>
<tr>
<td>IN5</td>
<td>I 1 N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
<td></td>
</tr>
<tr>
<td>ING</td>
<td>I 1 N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
<td></td>
</tr>
<tr>
<td>IPD</td>
<td>I 1 O</td>
<td>0</td>
<td>0</td>
<td>Indicates status of record in processing cycle</td>
<td></td>
</tr>
<tr>
<td>ISUPP</td>
<td>I 130</td>
<td>0</td>
<td>0</td>
<td>Supplemental sick pay</td>
<td></td>
</tr>
<tr>
<td>ITOH</td>
<td>I 1723</td>
<td>0</td>
<td>0</td>
<td>Account number for posting to in General Ledger</td>
<td></td>
</tr>
<tr>
<td>IWEK</td>
<td>I 5</td>
<td>1</td>
<td>1</td>
<td>Week of the month</td>
<td></td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/DEMP.</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWVA</td>
<td>I/N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>K</td>
<td>I/T</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Last card test</td>
</tr>
<tr>
<td>LAST</td>
<td>I/T</td>
<td>XXX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Last record number in file</td>
</tr>
<tr>
<td>LBO</td>
<td>I/N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LBT</td>
<td>I/N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LMC</td>
<td>I/N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LST</td>
<td>I/T</td>
<td>250 250</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Last record number in a file</td>
</tr>
<tr>
<td>LYRHR</td>
<td>I/O</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>This year's accumulation of hours worked for vacation pay</td>
</tr>
<tr>
<td>M</td>
<td>I/T</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Used in DO loop</td>
</tr>
<tr>
<td>MAR</td>
<td>I/I/O</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Marital status-(1-single),(2-married)</td>
</tr>
<tr>
<td>MUNC</td>
<td>I/N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>NADWH</td>
<td>I/O</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>Additional withholding amount</td>
</tr>
<tr>
<td>NAME</td>
<td>A/E/I</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Dummy area to allocate space for name</td>
</tr>
<tr>
<td>NCHCK</td>
<td>I/O</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>Check number used for this employee</td>
</tr>
<tr>
<td>NCU</td>
<td>I/I/O</td>
<td>XX.XX</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>Credit union deduction</td>
</tr>
<tr>
<td>NCUDDI</td>
<td>I/O</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>Monthly Credit Union deductions (in dimes)</td>
</tr>
<tr>
<td>NDUES</td>
<td>I/I/O</td>
<td>XX.XX</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>Union dues deduction</td>
</tr>
<tr>
<td>NINS</td>
<td>I/I/O</td>
<td>XX.XX</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>Insurance deduction</td>
</tr>
<tr>
<td>NMISC</td>
<td>I/O</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>Miscellaneous deductions</td>
</tr>
<tr>
<td>NPI</td>
<td>I/T</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Plant number</td>
</tr>
</tbody>
</table>

*Mode:  I = integer, R = real, D = decimal, A = alphabetic*
## VARIABLE SUMMARY SHEET

**Application**: PAYROLL SYSTEM  
**Date**: 8/5/67  
**Program Name**: File Create  
**Programmer**: [Name]  

### FUNCTION OF VARIABLES

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRATE</td>
<td>I</td>
<td>I.0</td>
<td>300</td>
<td>125</td>
<td>Employee pay rate</td>
<td></td>
</tr>
<tr>
<td>NSX</td>
<td>I</td>
<td>I.0</td>
<td>3</td>
<td>1</td>
<td>Sex: (1-female), (2-male), (3-trucker)</td>
<td></td>
</tr>
<tr>
<td>NSEN</td>
<td>I</td>
<td>3.0</td>
<td>Always 9 digits</td>
<td>Social Security number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSTAS</td>
<td>I</td>
<td>10</td>
<td>5</td>
<td>1</td>
<td>Employee status: (1-union), (2-trucker), (3-non-union, full time), (4-non-union, part time), (5-terminated)</td>
<td></td>
</tr>
<tr>
<td>NSTCK</td>
<td>I</td>
<td>2.0</td>
<td>xx.xx</td>
<td>Stock deduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSTKD</td>
<td>I</td>
<td>10</td>
<td>ø</td>
<td>ø</td>
<td>Monthly stock deductions</td>
<td></td>
</tr>
<tr>
<td>NUA</td>
<td>I</td>
<td>1.0</td>
<td>XXXX</td>
<td>ø</td>
<td>United appeal deductions</td>
<td></td>
</tr>
<tr>
<td>NUM</td>
<td>I</td>
<td>2.0</td>
<td>XXXX</td>
<td>1000</td>
<td>Clock number</td>
<td></td>
</tr>
<tr>
<td>NSWMP</td>
<td>I</td>
<td>10</td>
<td>ø</td>
<td>Number of weeks employed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMPD</td>
<td>I</td>
<td>10</td>
<td>ø</td>
<td>Number of weeks paid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMPF</td>
<td>I</td>
<td>10</td>
<td>17</td>
<td>Federal exemptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMPFS</td>
<td>I</td>
<td>10</td>
<td>17</td>
<td>State exemptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QRTD</td>
<td>R</td>
<td>%</td>
<td>0</td>
<td>0.00</td>
<td>Quarter to date information: (1) gross, (2) FIT, (3) FICA, (4) FICA wages, (5) sick pay, (6) social sec.</td>
<td></td>
</tr>
<tr>
<td>YTD</td>
<td>R</td>
<td>%</td>
<td>0</td>
<td>0.00</td>
<td>Year to date information: (1) gross, (2) FIT, (3) FICA, (4) FICA wages, (5) sick pay, (6) social sec., (7) loc. tax, (8) reg. hrs., (9) OT hrs., (10) bonus hrs., (11) reg. earns., (12) OT earns., (13) bonus earns.</td>
<td></td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
FORTRAN CODING FORM

Program: PAYROLL SYSTEM
Programmer: File Creation

Punching Instructions

<table>
<thead>
<tr>
<th>Statement Number</th>
<th>1</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>45</th>
<th>50</th>
<th>55</th>
<th>60</th>
<th>65</th>
<th>70</th>
<th>73</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement 1</td>
<td>C</td>
<td>J09.I NAME</td>
<td>PAYROLL SYSTEM</td>
<td>1</td>
<td>FILE</td>
<td>CREATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 2</td>
<td>C</td>
<td>JOB NUMBER</td>
<td>PAY1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 3</td>
<td>C</td>
<td>PROGRAMMER</td>
<td>C. R. K. L. I. C. K.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 4</td>
<td>C</td>
<td>DATE CODED</td>
<td>12/23/67</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 5</td>
<td>C</td>
<td>DATE UPDATED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 6</td>
<td>C</td>
<td>FILE</td>
<td>FILE</td>
<td>RECORD</td>
<td>NO.</td>
<td>01.</td>
<td>RECORDS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 7</td>
<td>C</td>
<td>INPUT FILES</td>
<td>2, COLFP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 8</td>
<td>C</td>
<td>OUTPUT FILES</td>
<td>2, WAPF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 9</td>
<td>C</td>
<td>3, MUCP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 10</td>
<td>C</td>
<td>4, LOAPF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 11</td>
<td>C</td>
<td>5, LFTP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 12</td>
<td>C</td>
<td>6, LCPF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 13</td>
<td>C</td>
<td>7, FINTO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 14</td>
<td>C</td>
<td>8, INDX1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 15</td>
<td>C</td>
<td>9, INDX2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement 16</td>
<td>C</td>
<td>10, INDX3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* A standard card form, IBM 888157, is available for punching source statements from this form.
**IBM FORTRAN CODING FORM**

<table>
<thead>
<tr>
<th>Program</th>
<th>Payroll System</th>
<th>Graphic</th>
<th>Punch</th>
<th>Card Form</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer</td>
<td>File Creation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### FORTRAN STATEMENT

#### Statement 1

```fortran
C----- C FOR COMMENT
C----- ALLOCATE ARRAY STORAGE
C----- INTEGER COM(I4)
C----- DIMENSION F1(REG(8)), INDEX(RF(15)), ISUF(RF(15)), ITOT(RF(11)), NAME(RF(9)),
C----- NSAI(RF(3)), QRTD(RF(6)), YTD(RF(14)),
C----- DEFINE THE FILES FOR THIS PROGRAM AS DESCRIBED ABOVE AND
C----- EQUIVALENCE THE VARIABLES FOR NEXT RECORD NUMBER.
C----- DEFINE FILE 1 (RS1, 168, U, I, COL), 2 (RS1, 168, U, I, IVA),
C----- 3 (RS1, 168, U, MUNC), 4 (RS1, 168, U, IBO),
C----- 5 (RS1, 168, U, LBT), 6 (RS1, 168, U, LMC), 7 (RS1, 168, U, IVA),
C----- 8 (RAI, 168, U, IN1), 9 (RAI, 168, U, IN2), 10 (RAI, 168, U, IN3),
C----- EQUIVALENCE (ICOI, IVA, MUNC, IBO, LBT, LMC)
C----- (IN1, IN2, IN3, IN4, IN5, IN6)
```

* A standard card form, IBM electro 888157, is available for punching source statements from this form.
<table>
<thead>
<tr>
<th>STATEMENT NUMBER</th>
<th>FORTRAN STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-4---</td>
<td></td>
</tr>
<tr>
<td>C-4--- INITIALIZED VARIABLES</td>
<td></td>
</tr>
<tr>
<td>C-4</td>
<td></td>
</tr>
<tr>
<td>C-4</td>
<td></td>
</tr>
<tr>
<td>C-4</td>
<td></td>
</tr>
<tr>
<td>6.8</td>
<td></td>
</tr>
<tr>
<td>6.8</td>
<td></td>
</tr>
<tr>
<td>6.8</td>
<td></td>
</tr>
<tr>
<td>6.8</td>
<td></td>
</tr>
<tr>
<td>6.8</td>
<td></td>
</tr>
</tbody>
</table>

* A standard card form, IBM electro 888157, is available for punching source statements from this form.
### FORTRAN CODING FORM

<table>
<thead>
<tr>
<th>Statement Number</th>
<th>FORTRAN Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NAWH: = 0</td>
</tr>
<tr>
<td>2</td>
<td>NCHGR: = 0</td>
</tr>
<tr>
<td>3</td>
<td>NWUD: = 0</td>
</tr>
<tr>
<td>4</td>
<td>NWSC: = 0</td>
</tr>
<tr>
<td>5</td>
<td>NSTD: = 0</td>
</tr>
<tr>
<td>6</td>
<td>NWMP: = 0</td>
</tr>
<tr>
<td>7</td>
<td>NWPDD: = 0</td>
</tr>
<tr>
<td>8</td>
<td>QRTD: (5) = 0</td>
</tr>
<tr>
<td>9</td>
<td>QRTD: (6) = 0</td>
</tr>
<tr>
<td>10</td>
<td>DO 69, M = 1: 14</td>
</tr>
<tr>
<td>11</td>
<td>YTD (M) = 0</td>
</tr>
</tbody>
</table>

---

### FORTRAN CODING FORM

<table>
<thead>
<tr>
<th>Statement Number</th>
<th>FORTRAN Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>READ PLANT NUMBER, WEEK NUMBER, AND CHECK NUMBER</td>
</tr>
<tr>
<td>6</td>
<td>READ (6, 4), NOPLIT</td>
</tr>
<tr>
<td>7</td>
<td>READ (6, 4), WEEK</td>
</tr>
<tr>
<td>8</td>
<td>READ (6, 5), CHECK</td>
</tr>
<tr>
<td>9</td>
<td>FORMAT (5, 11)</td>
</tr>
<tr>
<td>10</td>
<td>FORMAT (5, 2)</td>
</tr>
</tbody>
</table>
A standard card form, IBM electro 889157, is available for punching source statements from this form.
<table>
<thead>
<tr>
<th>Statement Number</th>
<th>FORTRAN Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 6</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>2, 7</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>3, 8</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>4, 9</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>5, 10</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>6, 11</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>7, 12</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>8, 13</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>9, 14</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>10, 15</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>11, 16</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>12, 17</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>13, 18</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>14, 19</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>15, 20</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>16, 21</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>17, 22</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>18, 23</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>19, 24</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>20, 25</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>21, 26</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>22, 27</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>23, 28</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>24, 29</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>25, 30</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>26, 31</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>27, 32</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>28, 33</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>29, 34</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>30, 35</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>31, 36</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>32, 37</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>33, 38</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>34, 39</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>35, 40</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>36, 41</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>37, 42</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>38, 43</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>39, 44</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>40, 45</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>41, 46</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>42, 47</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>43, 48</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>44, 49</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>45, 50</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>46, 51</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>47, 52</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>48, 53</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>49, 54</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>50, 55</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>51, 56</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>52, 57</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>53, 58</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>54, 59</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>55, 60</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>56, 61</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>57, 62</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>58, 63</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>59, 64</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>60, 65</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>61, 66</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>62, 67</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>63, 68</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>64, 69</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>65, 70</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>66, 71</td>
<td>C FOR COMMENT</td>
</tr>
<tr>
<td>67, 72</td>
<td>C FOR COMMENT</td>
</tr>
</tbody>
</table>

**FORTRAN STATEMENTS:**

1. **TABLE:**
   - **Program:** Payroll System
   - **Punching Instructions:**
   - **Page of Program:**

2. **FORTRAN STATEMENTS:**
   - C FOR COMMENT
   - READ ALL INFORMATION FOR ONE EMPLOYEE AND CHECK FOR LAST CARD
   - IF (k-9) (0, 100), 10
### FORTRAN CODING FORM

<table>
<thead>
<tr>
<th>Program</th>
<th>PAYROLL SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer</td>
<td>FILE CREATION</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Punching Instructions</th>
<th>Page of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphic</td>
<td>Card Form #</td>
</tr>
<tr>
<td>φ0112</td>
<td>Identification</td>
</tr>
<tr>
<td>Punch</td>
<td>PAYrol 60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FORTRAN STATEMENT NUMBER</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>(---)</td>
</tr>
<tr>
<td>35</td>
<td>(---)</td>
</tr>
<tr>
<td>40</td>
<td>(---)</td>
</tr>
<tr>
<td>45</td>
<td>(---)</td>
</tr>
<tr>
<td>50</td>
<td>(---)</td>
</tr>
<tr>
<td>55</td>
<td>(---)</td>
</tr>
<tr>
<td>60</td>
<td>(---)</td>
</tr>
<tr>
<td>65</td>
<td>(---)</td>
</tr>
<tr>
<td>70</td>
<td>(---)</td>
</tr>
<tr>
<td>72</td>
<td>(---)</td>
</tr>
</tbody>
</table>

---

C FOR COMMENT

---

C SETUP EMPLOYEE STATUS CODE, STATE EXEMPTIONS, AND Q-T-A INFORMATION

---

\( 
\begin{align*}
\text{M.TAIS} &= 1 \\
\text{N.XMPS} &= \text{N.XMPF} \\
\text{QRTD}_1 &= \text{YTD}_1 \\
\text{QRTD}_2 &= \text{YTD}_2 \\
\text{QRTD}_3 &= \text{YTD}_3 \\
\text{QRTD}_4 &= \text{YTD}_4 \\
\end{align*} \\
---

---
A standard card form, IBM Form 888307, is available for punching source statements from this form.
**FORTRAN CODING FORM**

<table>
<thead>
<tr>
<th>Statement Number</th>
<th>FORTRAN Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CREATE THE INDEX ENTRY FOR THIS EMPLOYEE AND WRITE HIS RECORD.</td>
</tr>
<tr>
<td></td>
<td>ONTO THE DISK. THEN GO BACK TO THE READ STATEMENT TO GET INFORMATION ON THE NEXT EMPLOYEE.</td>
</tr>
<tr>
<td></td>
<td>INDEX(TICAL) = NUM.</td>
</tr>
<tr>
<td></td>
<td>WRITE TO THE DISK.</td>
</tr>
<tr>
<td></td>
<td>WRITE(NOALT 'TICAL', NUM, NAME, INSSAM, NSTAS, MDRVS, NUMP, NUMD,</td>
</tr>
<tr>
<td></td>
<td>MGR, NEMP, NXAPS, NSEX, NMATE, YTD, QTD,</td>
</tr>
<tr>
<td></td>
<td>LTHL, NCN, NEUDD, NCHCR, NADWKO, NSTCK, MINS,</td>
</tr>
<tr>
<td></td>
<td>NMISC, HUA, NSTD, ISWPP, INITY, FPD.</td>
</tr>
<tr>
<td></td>
<td>GO BACK FOR ANOTHER EMPLOYEE'S INFORMATION.</td>
</tr>
<tr>
<td></td>
<td>GO TO SGP.</td>
</tr>
</tbody>
</table>

* A standard card form, IBM electro 888157, is available for punching source statements from this form.
IBM FORTRAN CODING FORM

<table>
<thead>
<tr>
<th>Statement Number</th>
<th>FORTRAN Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C--- LAST CARD HAS BEEN READ.</td>
</tr>
<tr>
<td>2</td>
<td>C--- INITIALIZE THE TRADE ASSOCIATION INFORMATION.</td>
</tr>
<tr>
<td>3</td>
<td>600 DO 610 IS=1,7.</td>
</tr>
<tr>
<td>4</td>
<td>650 FIBBIE(X)=6.</td>
</tr>
<tr>
<td>5</td>
<td>C--- WRITE THE INDEX OF EMPLOYEES FOR THIS PLAN TO DISK.</td>
</tr>
<tr>
<td>6</td>
<td>LAST=ICOL = 1.</td>
</tr>
<tr>
<td>7</td>
<td>WRITE (20,N,1) (INDEX(I),X=LAST).</td>
</tr>
<tr>
<td>Statement Number</td>
<td>FORTRAN Statement</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td></td>
<td>C-——— WRITE THE RECORD FOR THIS PLANT TO DISK, THE NUMBER OF EMPLOYEES.</td>
</tr>
<tr>
<td></td>
<td>C-——— IN THE PLANT TO THE INDEX AND STOP.</td>
</tr>
<tr>
<td></td>
<td>C-——— WRITE(1,'NODLT') COMP, TCHECK, TWEERK, TIRE, IDOT, ISKMAX.</td>
</tr>
<tr>
<td></td>
<td>C-——— WRITE(IND'LST), LAST.</td>
</tr>
<tr>
<td></td>
<td>C-——— CALL EXIT.</td>
</tr>
<tr>
<td></td>
<td>C-—— END.</td>
</tr>
</tbody>
</table>
Example 2: Add Name to the File

This program is an extension of PAY01, File Creation. Because the employee record contains more than 80 characters, one card is not sufficient. The name field for each employee appears on a second card which is processed by this program.

The dummy name field, set up in the disk record by PAY 01, is filled in with the actual name on the card.

The program illustrates a simple single-file update with no calculations. The following programming techniques have been used (see Section 35 for a listing of this program):

1. Updating masters. The master file is updated by changing the name field in the master record. Note that only the variable name in the output list has been changed.

2. Searching an Index. The index to the file contains an entry for each employee. The clock number is placed in the index at a location corresponding to the record number for the employee. Each index entry is examined to find a match with the clock number on the card (statements 120-125). When a match is found, the location of the match in the index is the employee record number in the disk file.

3. Indicating exceptional conditions. When the index is searched, it is possible that the clock number on the card will not match any index entry. If this occurs, the clock number is printed in the following message:

   CLOCK NO XXXX NOT IN FILE.
Example 3: Changes to the File

This program illustrates a complex single-file update procedure. Any one of 16 different changes can be performed.

The master file is the file created by PAY01 and PAY02. The transaction file is on cards, where each card contains the clock number, a code indicating where the change should be applied, and the new or changed information.

The one important change this program will not perform is deletions from the file. However, this may be accomplished by changing the pay rate to zero.

The following programming techniques should be noted in this program (see Section 35 for a listing of this program):

1. Setting a switch rather than testing. The change code is a two-digit number from 01 to 16 (statements 105+1 and 106). When it has been validated, proven greater than zero and less than 16, the code is used as the index for a computed GO TO statement (statement 140). This saves the program a set of IF statements, each statement testing the code and deciding on an action.

2. Detailed data validation. Since PAY01 and PAY02 were so careful about building the file and making sure the data was correct, common sense indicates that the same care should be extended to any changes to it. This is done through checks, not only on the change code, but on the plant number, the clock number, and, where applicable, the change itself. Note that the addition to the file of a new employee causes a check to see whether that employee clock number is already in the index.

3. Use of the alternate stacker. Any time an error is detected, the card involved in the error is selected to the alternate stacker of the IBM 1442 (statements 3 + 1, 8 + 1, 5 + 1, and 7 + 1). This will save the operator the task of picking out those cards with errors.
Example 4: Calculations and Payroll Register

This program consists of extensive calculations and report writing. Payroll calculations are performed, including calculations of gross pay, taxes, voluntary deductions, and net pay. The report shown is the payroll register.

In addition, the calculations are balanced to control totals and each disk record is extended with the current period's calculations.

The following programming techniques have been used (see Section 35 for program listing):

1. Arithmetic Statement Function. Since the 1130 is a binary computer, decimal fractions may not be expressed exactly in binary. This inaccuracy may be avoided by performing all calculations with whole numbers. (See Section 70.10.20.) When calculations are complete, the result must be half-adjusted and the decimal point placed. Since there are many calculations in this program, it makes sense that the rounding procedure should be set up only once and accessed from many different places. The Arithmetic Statement Function, PHIL, will be used to do this.

2. Use of data switches. Since the check number, week number, and maximum check amount are not permanent, a facility must be built into the system to change them. By setting the console data switches appropriately (statements 3 + 5 and 71), each or all of these numbers can be changed. A hard-copy record of any changes is produced on the console printer.

3. Zero balance test. The control totals are compared with accumulations produced during the processing of the file. The original control totals, the accumulated totals, and the differences are printed. If the differences are not zero, the operator knows that further examination of the output is necessary. (See statements 15-18.)

4. A variety of calculations. The calculations performed with this program are more extensive than the other sample programs. The first set of calculations is used to initialize the program variables, while the second set initializes the plant variables. The third set initializes the variables for an individual.

The remaining detail calculations pertain to regular, overtime, and bonus earnings, taxes (including federal, state, and local), and voluntary deductions. Finally, the net amount is calculated and plant totals are accumulated.

5. Backup is built into the system. To provide a means of recovery when an error condition or an out-of-balance condition occurs, the calculated information (gross, net, tax, etc.) is punched into the employee's weekly card (see statement 9). A simple list of these cards will thus supply sufficient information to check or reconstruct portions of the file.

6. Another type of half-adjusting. In printing the payroll register the dollar and cents figures should appear with decimal points. To round off, reposition the decimal point, and clear fractions, the WHOLE Function (from the Commercial Subroutine Package) is used (see statements 515-515 + 11).

\[
AMT = \text{WHOLE} \left( AMT + \left( AMT/\text{ABS}(AMT) \right) \times 0.5 \right) / 100.
\]
Example 5: Check Writing

This program demonstrates the use of the Commercial Subroutine Package (CSP) in preparing a report—namely, the check and check stub.

In this example, the employee file is accessed sequentially. If the paid indicator is set appropriately, a check is written. In either case, the next employee record is read.

Control totals are carried, and a zero-balance check is performed.

The following programming techniques should be noted in this program (see Section 35):

1. The use of subroutines. There are three specific operations which are used many times (see statements 91 + 9 - 95 + 5). These are PUT, MOVE, and EDIT. PUT converts from real format to A1 format, MOVE moves information, and EDIT inserts and removes characters. Rather than repeating the statements that perform these three operations each time, it is much simpler and shorter to make subroutines out of the statements. This, in addition to saving core storage, is much easier to test and document. All three subroutines are supplied with the 1130 Commercial Subroutine Package.

2. Editing data for output. The use of the EDIT subroutine is a very powerful technique. It requires two kinds of data. The first is the data to be edited, and the second is a description of the result, the edit mask. As can be seen, the edit mask is treated as a constant and is initialized at the beginning of the program (see statement 4). The result of editing can be seen in the amount field of the check specimen shown.
Example 6: Check Register

This program illustrates a report in which detailed items are written, three up, (three items per line). A plant file is accessed for each employee (see statement 655), and a line containing check number, employee clock number, employee name, and net check amount is composed. When three employees have been placed on one line, the line is printed.

This technique will produce a very concise report, easily read, filed, and used. The technique also decreases printer time to produce the report by decreasing the number of lines to be printed.
Example 7: 941 Report

This program will process multiple files to produce the 941 report. One report is produced for each plant.

Note that a count of the lines printed on each page is kept (see statements 195 + 5 and 150). In this way, headings can be and are printed at the top of each page in the report.

Also, notice that the plant totals are reset only when a new plant is to be processed (see statements 2 + 1 thru 3 - 2), while page totals are reset when a new page is to be printed (see statements 4 + 4, 4 + 5).
Section 30: TESTING EFFECTIVELY

CONTENTS

Introduction ........................................... 30.01.00  
Testing Strategy ...................................... 30.10.00  
Testing Tactics ...................................... 30.20.00  

Testing Hints ....................................... 30.30.00  
Summary .............................................. 30.40.00
INTRODUCTION

Now that programming is "finished", it is time to evaluate what you have in relation to your objectives. Do your programs and systems produce the results you want? To find this out you must test the programs -- but not until you have a plan.

Experience has shown that more time can be wasted in testing than has originally been allotted to testing.

In other words, testing must be performed effectively.

* * * * *

There is a great temptation to get a newly written program on a machine and see it run. A little extra effort before going to the machine can save a great deal of time and effort in the long run. If you must travel some distance to test a program before the installation of your own machine, it also means real money saving.

The chances are very good (99.99%) that your program contains errors of various types:

1. Programmer clerical errors. It is easy to make minor clerical errors when filling out the coding sheets. Although they are minor, the program will not work properly until they are corrected.

2. Programmer procedural errors. The number of procedural errors will depend on the experience and proficiency of the programmer. These are caused by not adhering to the programming rules as outlined in the language manuals.

3. Card punch errors. Errors may be introduced when the program is punched into cards. Punching programs into cards is very exacting, and the keypunch operator must be very careful. Because of the nature of the information on the program sheets, it is difficult to achieve speed and accuracy in punching.

4. Program logic errors. Logic errors may be caused by poor or incomplete analysis of the problem prior to programming, or by incorrect programming after correct analysis. In any event the program must be able to either process, or properly reject, all the various pieces of information that will be given. Someone who is intimately familiar with the procedure, as it is now being done, should review the logic of the program for completeness.

Most clerical errors, both programming and card punching, can be detected by a careful review of the material. Key verification should always be done, and it is essential to proofread coding sheets before they are punched. The most common errors occur in the use of 0 and O, 1 and I, 2 and Z. Standards should be adopted in which, for instance, alphabetic O is written O, zero is , Z as , I as a printed capital (I), and 1 as a straight line (1). It is wise to formally familiarize your keypunch operators with the adopted conventions.

Program procedural errors can often be detected by having someone other than the original programmer review the programming sheets. Even where the programmers are relatively inexperienced, they will often catch errors in syntax (grammar of forming statements). This review can also serve as an excellent way to improve programmer knowledge.

During a review of the program, program logic errors are more difficult to catch. This is particularly true when the person who is familiar with the procedure is not also familiar with programming. Logic errors are generally caught during testing when sample data is processed by the program. The sample data must be prepared so that all of the various exceptions, combinations, and ranges of information are introduced to the program, insofar as it is practical to do so. It should be remembered that any element or combination of elements that is not tested is very likely to appear eventually; if it can happen, it will.

At the time that your program is assembled or compiled on the system you are installing, a series of diagnostic tests is also made to detect many of the potential errors, and these errors are noted.

By properly prechecking your programs, you can materially reduce the amount of time to get a program compiled and tested successfully. Care in the preparation of test data will also detect logic errors so that they can be corrected before the processing of actual data.

The final test of any program is the successful processing of "live" data, after which the results can be compared against those obtained by the previous system.

Note: If the results of this last test do not agree with previous results, check again to be sure what the right answers should be. Sometimes the old system has not produced the correct solutions.
TESTING STRATEGY

Any good system is like a successful athletic team. Each member must do his job well, and all members must work together. These two things are what you must accomplish with your testing strategy. Each individual program is tested. When all programs give correct results, pairs are tested. When the first pair gives correct results, another run is added to the system. Finally, all runs are tested together, and the entire system is checked out.

The individual tests are the foundation of the system's test. A deck of test cards should be made up for each program (or subprogram) and kept for use in testing the program again in the future.

The ideal rule to follow in deciding what test data to include is this: include every field at least once under every condition in which it can occur, not only by itself, but with every possible combination of conditions in which all the other fields can occur. With a simple program this is easy enough to do, but where many fields appear under many conditions, the number of possible combinations can become enormous. Then your programmer must use his judgment in making up a limited set of test cards that covers the possibilities adequately.

The test cards should be created, then listed. For each set of test data, a "prediction" of the results that will appear on the output forms or cards should be made. Then, when actual testing is performed, your programmer cannot be easily misled into believing that his output is correct when it is not.

The first data in the test deck should test only the ordinary, easiest, most straightforward conditions. Next, multiple conditions can be combined on one card or record. Finally, error conditions can be tried. The reason for this careful progression is that unless the simple situations are proved first, it is possible to spend many hours trying to determine which of several possible causes for a "bug" is the true one.

Avoid setting up your tests in such a way that you count on the output of one program to act as input to another. Have at least one independent set of test data for each program you are testing. "Merged" or "linked" testing is a valuable means of proving a system's overall validity, but it should not be done until each program is individually tested.

After a successful test, both the test input and output should be retained, as part of program documentation, to make future testing easier. Also, when testing program modifications, test not only the modifications but the entire program. In other words, your sample test data should expand with each modification, so that the entire system may be tested at any time.
TESTING TACTICS

Many techniques exist to assist your programmer during the checkout phase of a program. Each has its own advantages and disadvantages. The one to be used for a particular problem will depend on your programmer's thoughts as to what area of his program is in error. Some very useful techniques are:

1. **Core Storage Dump.** This is a printout of the contents of core storage. There are two methods of producing it.

   The first is with one of the utility programs supplied with the 1130 Programming Systems. These utilities will produce a core storage dump in hexadecimal.

   Since manual hexadecimal-to-decimal conversion is very tedious and time-consuming, this method is not recommended.

   The recommended method of dumping core storage is with the dynamic DUMP facilities of FORTRAN and the Assembler Language. The information dumped with this method can appear in hexadecimal, integer, or real format.

   In FORTRAN, the DUMP facility is accessed through use of the PDUMP subroutine. You would write CALL PDUMP (A1, B1, F1, ..., An, Bn, Fn).

   Blocks of core storage are dumped. A1 and B1 are variable data names, subscripted or nonsubscripted, indicating the inclusive limits of the first block of storage to be dumped. Similarly, An and Bn indicate the inclusive limits of the nth block of storage to be dumped.

   The format of a block is determined by the Fx associated with that block. F1 through Fn are integers and are assigned in the following manner:
   
   0 = Hexadecimal
   4 = Integer
   5 = Real

   The Assembler Language dump facilities, DUMP and PDMP, are used in a similar fashion.

   All of the core storage dump facilities will produce a printout of core storage, by address. You should use these facilities when a program "bug" requires, in the judgment of your programmer, an examination of all or part of core storage.

2. **Arithmetic Trace.** The use of this technique involves subroutines that are executed whenever a value is assigned to a variable on the left of an equal sign. If Console Entry Switch 15 is turned on at execution time, and the *ARITHMETIC TRACE FORTRAN control record is used, the value of the assigned variable is printed, as it is calculated, with one leading asterisk.

As an optional use, you can elect to trace only selected parts of the program by placing statements in the source program to start and stop tracing. This is done as follows:

   CALL TSTOP (to stop tracing)
   CALL TSTRT (to start tracing)

   Thus, tracing occurs only if:
   
   - The trace control record is compiled with the source program.
   - Console Entry Switch 15 is on (can be turned off at any time).
   - A CALL TSTOP has not been executed, or a CALL TSTRT has been executed since the last CALL TSTOP.

   If tracing is requested, an *IOCS control record must also be present to indicate that either typewriter or printer is needed. If both typewriter and printer are indicated in the *IOCS record, the printer is used for tracing.

   Use of this facility will increase execution time considerably. The trace facility should not be present in a production program; if it is, you should recompile the production program after testing is complete, leaving out the trace.

3. **Transfer Trace.** In this case, the FORTRAN compiler generates linkage to trace routines which are executed whenever an IF statement or Computed GO TO statement is encountered. If Console Entry Switch 15 is turned on at execution time and the *TRANSFER TRACE FORTRAN control record is used, the value of the IF expression or the value of the Computed GO TO index is printed. For the expression of an IF statement, the traced value is printed with two leading asterisks. The traced value for the index of a Computed GO TO statement is printed with three leading asterisks.

   The optional use of trace explained under Arithmetic Trace also applies to Transfer Trace (use of TSTOP and TSTRT), as does the information following optional use.

4. **Extensive Use of PAUSE.** It may turn out that some parts of your program execute correctly and some incorrectly. What you would like to do is to check the progress of the program while it is running. A very useful technique is to place PAUSEs at strategic places throughout your program. In order to know where the program is at any point in time, number the PAUSEs consecutively:

   C------READ INPUT
   PAUSE 1
   CALL READ(IN, 1, 80, N)
   C------IDENTIFY INPUT
   PAUSE 2
IF(IN(22)-1) 3, 4, 5
3 CALL MOVE(IN, 1, 27, IWK, 1)
C-----TYPE ZERO CARD
PAUSE 3
etc.

The PAUSE number will be displayed in the accumulator. Use of this technique will let you follow the logic of the program (IFs and GO TOs) without severely slowing its execution.

5. Additional Print Lines. This technique is sometimes called "selective tracing". Again, rather than severely slowing the execution of a program and printing the result of every replacement operation, only selected variables and/or fields will be printed. Use of the FORTRAN WRITE statement or the 1130 Commercial Subroutine Package PRINT subroutine will allow you to follow the progress of variables and/or fields as their contents change during program execution.

6. Console Debugging. This technique should be used only as a last resort. It involves manual inquiry into the system via the console switches, dials, and keys. In most cases, the previously mentioned techniques will provide you with all the information necessary to debug.
TESTING HINTS

1. To test the logic in a program that uses Commercial Subroutine Package I/O, use standard FORTRAN READ and WRITE for I/O. This makes the trace facility available. When finished, use Commercial Subroutines READ and PRINT for overlapped I/O.

2. Ask yourself: What must be done to re-create information if the disk cartridge is lost? How long will it take?

3. Keep testing in mind when planning the development of various runs. That is, write the file creation and maintenance programs before the report programs that use the files.
SUMMARY

If program testing techniques are properly planned, a minimum amount of machine time is consumed during program checkout. Manual inquiry into a system via the console is extremely expensive in machine and operator cost; little is learned in return for dollars expended. Time spent desk checking is well invested, since most of the logical errors may be detected before the program actually enters the computer testing phase.

In trying to make the maximum number of runs during a test session, your programmer may be tempted to make rapid patches without pausing to annotate such changes thoroughly. Such urgency is seldom fruitful in the end.

The program testing phase should be carefully and thoroughly planned, executed, and documented. The following checklist should be used as a guide to ensure maximum productivity for program testing.

After coding the program and preparing revised flowcharts and other supporting documentation, the testing procedure begins.

1. Prepare test data and precalculate results.
2. Punch and verify program cards.
3. List program cards.
4. Desk-check the program. Look for:
   a. Errors in logic
   b. Endless loops
   c. Incorrect use of program switches
   d. Unsatisfied or incomplete coding for the problem definition
   e. Inefficient program (time and storage)
   f. Incorrect data field lengths
   g. Improperly signed fields
   h. A name for each variable
   i. Improper indexing
   j. Initialization of routines and storage
   k. Duplicate names
   l. Misspelling and punching errors
   m. Invalid operations
   n. Necessary control cards
   o. Improper alignment of card columns
5. Manually simulate the computer process using test data.
6. Compile the program.
7. Perform error analysis with error listing and program printout.
8. Correct the program.
   a. Card programs. Correct the source deck and recompile the program. To facilitate card handling with object decks, label the object deck with a marking pen. The first and last card of the object deck should be so labeled. The top edge of all such cards may also be marked.
   b. Disk programs. When the program is prepared on disk, corrections are made to the source deck. This is accomplished by placing the corrections in the source deck and then recompiling and restoring the program. Alter the program listing and update the program flowchart to reflect source deck corrections.
9. Prepare detailed instructions for machine operation during the test session.
10. Pre-test-session familiarization.
   a. Console operation
   b. Input/output devices
   c. IOCS
   d. Utility routines such as clear storage and load programs, file generators, trace programs, storage and disk print programs, sort and merge programs, and check point and restart programs.
11. Test documentation and materials. To reduce confusion, all materials should be clearly labeled with the name of your organization, program name, content, and date. Each person should have a list of items for which he is responsible:
   a. Program flowcharts
   b. Compiled program listings
   c. Test data decks and disks with test data listing (a duplicate copy may be desirable)
   d. Precalculated results of test data and listing of expected output with each test case
   e. Card and disk record layouts
   f. Internal storage map
   g. Printer carriage control tape
   h. Operator checklist, providing all the information the operator needs to set up the data processing system for the running of each program:
(1) Job or program name
(2) Operation name
(3) Machine setup
   (a) Disk cartridge assignment
   (b) Input cards or tapes
   (c) Output cards or tapes
   (d) Carriage tape
   (e) Sense switch settings
(4) The sequence of events to run the test
(5) Listing of all possible messages and halts
(6) Switch and index listings
(7) List of paper forms or card stock for auxiliary equipment
i. Object deck or disk cartridge
j. Blank forms, cards, disks
k. Source deck and listings
12. The test session
a. Plan the test session in advance. Decide upon the sequence in which programs shall be tested. Programs should take precedence in testing according to their importance, and the most important programs should be re-tested as often as possible until they are completely debugged. Schedule a workload greater than can be accomplished in the allotted test time. Assign duties (such as handling the card reader, punch, printer, disk cartridges, and console) to each person attending.

b. Arrive early. Confirm the testing schedule that was established in advance of actual testing. This schedule may best be laid out as a series of half-hour to full-hour sessions with one- to two-hour breaks in between.
c. Be familiar with the latest versions of all programming systems to be used.
d. Make certain that the test packet is organized properly. Test the higher-priority and larger programs first. Each program should have its own input test data; one program should not be dependent on another program that was run earlier in the same session.
e. Make sure that all units are in the proper initial status--for example, printer restored, disk units ready, no leftover cards in the reader or punch.

f. Debugging at the console is time-consuming, error-prone, and generally nonproductive. When the program hangs up, the following steps should be taken immediately:
   (1) Note the console status—indicators, lights and registers.
   (2) Take core storage dumps.
   (3) Take disk dumps.
   (4) Go on to next program or cease work.

Even if a program goes to end-of-job and appears correct, the above steps should be taken in order to simplify correcting errors discovered later. When a program hangs up, do not force it to continue without taking down status information, since the conditions causing the original hangup would then be destroyed.

g. Label all core storage dumps, disk dumps, console sheets, etc., with date, time, and program identification.
h. Debug off the console with deliberate speed. With the above items, there is more information to aid in locating the reason for the hangup than is available at the console. Do not make hurried corrections to a program in a false effort to maximize usage of test time. Do not, however, spend three hours at a desk to save five minutes on the system. Strive for a reasonable cost balance.

Before testing the program again, find all possible bugs, not just the one that caused the hangup. Step further through the program after each test to ensure that the program will not hang up on the next instruction or routine. Correct all errors in output content and format. Strive for perfect output from each test.
i. Note all corrections on the program listing. Corrections that affect the halt, switch, or index listings should be updated accordingly.
j. Note the reason for the correction adjacent to the card itself. Be sure to include number and date. A post-test listing of cards is desirable for reference when correcting the source deck.
k. Generate a new program listing after an appropriate number of cards have been added to the program. Update the program flowchart to reflect the current status of the program.

1. Keep documentation current. This eliminates the waste of time and effort trying to pick up changes during testing or debugging.

13. Post-test evaluation. Every test session should be followed by a thorough evaluation:
   a. Was the pretest preparation adequate?
   b. Were there any areas of preparation that could be improved to yield a more effective test?
   c. Were there areas of preparation in which you spent too little?
   d. Did the test point up any areas of weakness in the coding? If so, are these types of errors documented so that stronger emphasis can be placed on them during future coding and desk checking?

   e. Was each machine session used effectively?
   f. Are there any corrections to the testing techniques that would make the next test more fruitful?
   g. What is the status of each program tested?
      (1) Is it completely tested? That is, has every program loop been tested, and do you have any reservations about calling this program complete?
      (2) Is it tested to the stage where the only changes left are in spacing and editing of the output data?
      (3) Are there logic errors left in this program?
   h. Did the test session achieve its objectives? If not, what adjustments in present scheduling are necessary?
Section 35: PROGRAM DOCUMENTATION

CONTENTS

<table>
<thead>
<tr>
<th>Introduction</th>
<th>35.01.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation Manuals</td>
<td>35.10.00</td>
</tr>
<tr>
<td>Program Information Manual</td>
<td>35.10.10</td>
</tr>
<tr>
<td>Operation Manual</td>
<td>35.10.20</td>
</tr>
<tr>
<td>Documentation Examples</td>
<td>35.20.00</td>
</tr>
<tr>
<td>Payroll System — Program</td>
<td></td>
</tr>
<tr>
<td>Information Manual</td>
<td>35.20.10</td>
</tr>
<tr>
<td>Payroll System — Operation Manual</td>
<td>35.20.20</td>
</tr>
</tbody>
</table>
INTRODUCTION

The final step in your installation program is to document everything you have done. Let us quickly review the importance of adequate documentation before discussing the form that your documentation may take.

The package of materials describing each program will become:
1. A source of information for implementing future changes.
2. An education device for familiarizing new operators and management personnel with the procedures.
3. A means of describing control procedures to your auditors.

It is a modern but well proven adage that a well documented installation is a sure sign of a smooth-running operation.

You should develop two manuals: the program information manual and the operation manual. Your basic library will consist of these two manuals together with this 1130 User's Guide, physical planning manual, the 1130 functional characteristics manual, the programming system reference manuals for FORTRAN and Assembler Language, the machine reference manuals for the I/O units you have ordered, and operating procedures manual for FORTRAN, Assembler Language, and Disk Monitor System. If you use the Commercial Subroutine Package, you will also want reference manuals and operating procedures manuals for that system. Consult the 1130 SRL Bibliography for descriptions and form numbers of the manuals, and for information about other IBM publications that provide further details on the subjects covered in this guide.
INSTALLATION MANUALS

Program Information Manual

Each application should have its own binder, which will be used by management, systems analyst, or programmer, and will contain:

1. **Job description.** This is the same for all programs with a job or application. It is a brief abstract.

2. **System flowchart.** This is also the same for all programs within an application, and shows how each program fits into the larger picture.

3. **Record layouts.** All record formats for the application are shown.

The three items above appear once for the application, whereas the items below are necessary for each program (you may want to place dividers, labeled with the program names, in front of each group of these):

1. **Form layout.**

2. **Variable Summary Sheet.** The purpose for which program variables are used is apparent at the time of writing, but again, as with program logic (of which variables are an integral part), the programmer rapidly forgets how he used them. The Variable Summary Sheet (see Section 25) will serve as a testing and program modification aid.

3. **Program flowchart.** Experience has proved that logic which is clear to the programmer at the time of writing is difficult to recall a short time later. The logic must, therefore, be documented in such a manner that testing will be accomplished in a minimum amount of machine time. Well documented logic is also valuable when the program is changed from time to time, either by the author or by another programmer who may be completely unfamiliar with it.

4. **Coding sheets or program listing.** To avoid confusion, the coding sheets should be discarded after the program listing is produced.

5. **Test data listing.** Test data should be listed and retained. As changes to the program are made, they may unintentionally affect parts of the original program. All original test data, therefore, along with any additional test data necessary for the change, should be processed to ensure that the program is operating properly.

6. **Test output.** This includes sample reports or cards, as produced by the test data.

7. **Machine setup sheet.** This is a guide to the operator, describing machine setup, source of input, disposition of output, and actions to be taken at machine halts.

8. **Detailed program flowcharts.** These must be included if the programmer is using Assembler Language. Since programs written in Assembler Language are not as easily read, or as clearly related to the job as FORTRAN programs, it is vital that your programmer draw a detailed program flowchart that carefully documents the program steps he has taken. Each block should cover only a few program steps, and should be cross-referenced to the program. It is advisable in most cases to include a general program flowchart, which provides a quick means of introduction to the logic and is exploded by the detailed flowchart.
Operation Manual

Intended for use by the operator, the operation manual is arranged so that each application has its own section. Usually, these materials are all kept in one book, at the 1130 console. In addition to the materials suggested below, the operation manual should include a copy of the operating procedures manuals supplied by IBM for the programming system being used.

Dividers of two kinds should be used: one for applications and one for programs within applications.

Behind each application divider should be a job description followed by a system flowchart of the entire application.

Behind each program divider should be all instructions to the operator. These may include (1) procedures to be followed to accomplish accounting controls, such as recording totals on a control sheet, checking critical items, and noting cross-footing messages, (2) recovery procedures -- that is, procedures for reconstructing or continuing a run that has been interrupted as a result of an operator, machine, or program error, (3) initial switch settings and their meaning, (4) halts, error messages and their meaning, and (5) I/O considerations.
DOCUMENTATION EXAMPLES

The examples in this section show the necessary documentation for those runs in the Payroll System which were coded under Section 25. Note that these examples are illustrations and, therefore, may not be considered complete, usable programs.
<table>
<thead>
<tr>
<th>Section</th>
<th>Subsections</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAY05: Check Writing</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Check</td>
<td>96</td>
</tr>
<tr>
<td>Variable Summary Sheet</td>
<td>97</td>
</tr>
<tr>
<td>PAY05 General Program Flowchart</td>
<td>103</td>
</tr>
<tr>
<td>Program Listing</td>
<td>104</td>
</tr>
<tr>
<td>Test Data Listing</td>
<td>111</td>
</tr>
<tr>
<td>Test Output</td>
<td>112</td>
</tr>
<tr>
<td>Machine Setup Sheet</td>
<td>113</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAY06: Check Register</th>
<th>114</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Check Register</td>
<td>114</td>
</tr>
<tr>
<td>Variable Summary Sheet</td>
<td>115</td>
</tr>
<tr>
<td>PAY06 General Program Flowchart</td>
<td>121</td>
</tr>
<tr>
<td>Program Listing</td>
<td>122</td>
</tr>
<tr>
<td>Test Data Listing</td>
<td>126</td>
</tr>
<tr>
<td>Test Output</td>
<td>126</td>
</tr>
<tr>
<td>Machine Setup Sheet</td>
<td>127</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAY09: 941 Report</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample 941 Report</td>
<td>128</td>
</tr>
<tr>
<td>Variable Summary Sheet</td>
<td>129</td>
</tr>
<tr>
<td>PAY09 General Program Flowchart</td>
<td>133</td>
</tr>
<tr>
<td>Program Listing</td>
<td>139</td>
</tr>
<tr>
<td>Test Data Listing</td>
<td>140</td>
</tr>
<tr>
<td>Test Output</td>
<td>140</td>
</tr>
<tr>
<td>Machine Setup Sheet</td>
<td>141</td>
</tr>
</tbody>
</table>
PAYROLL APPLICATION

JOB DESCRIPTION

The Payroll System is composed of 16 different runs. From the source documents, produced at the six plant sites, cards are punched. These cards are used to store the payroll information on the disk cartridge.

At this point the system uses cards only for transition between jobs. The input data, employee records, is read from the disk and updated before being written back. This gives a highly flexible system, in which I/O, because of the disk, is very fast.

The system produces the following reports:
- Checks and check stubs
- Check register
- Payroll register
- Deduction registers for
  1. Union dues
  2. Credit union
  3. Stock
- 941 quarterly report

SYSTEM FLOWCHART

Narrative

The system consists of 16 programs. The Files Creation program is first. Data decks are keypunched for each individual, in sets, by plant. The data is edited and, when correct, loaded on the disk by PAY01. Three files are created: a master file, an index file, and a plant information file. A second data deck with employee clock number and name is loaded onto the master file by PAY02.

Changes to the disk information are made by PAY03. Documents, received from personnel departments at the individual plants, are checked, summarized, keypunched, and verified. Time sheets, submitted by the plant payroll departments, are keypunched and verified. All of these cards are processed by PAY16, which edits and generates control totals. PAY04 then processes these cards, performing all payroll calculations. Cards are read, pay computed, disk files updated, and cards extended with current pay figures. After all cards are processed, a payroll register is printed.

Checks are printed by PAY05. A header card is read and the checks are printed from the disk file. PAY06 lists the check register from the disk file. In the event of an error in computing pay, PAY11 provides the means of voiding checks. The extended time cards from PAY04 are read in and the affected employee records are reset. The above are weekly runs.

At month end, registers are prepared showing each individual's deductions for the month: PAY13 writes union dues register.
PAY14 writes credit union register.
PAY15 writes stock deductions register.
PAY12 resets charity deductions code.

At the end of the quarter and at the end of the year PAY07 and PAY08 are used to balance the disk files to control totals.
PAY09 produces the 941 tax report.
PAY10 produces a tax worksheet used to determine tax reliability.
At the present time the program for W2 reports has not been written.
Disk Payroll File

Control Totals

CALCULATED PAY 05 PAYROLL CHECKS

Pay Checks and Stubs

Total on Adding Machine

TAPE

B

C

Balance to Totals; If Incorrect, Go to D

Only When Totals Balance

Burst, Sign and Distribute Paychecks and Stubs

Disk Payroll File

PAY 06 CHECK REGISTER

Check Register

Balance to Totals; If Incorrect, Go to E

Distribute Check Register

Disk Payroll File

Store

File D

Control Totals

Section | Subsections | Page
---|---|---
35 | 20 | 10

09
Only When Totals Do Not Balance

Destroy Checks

Disk Payroll File

Control Totals

File D

PAY 11 VOID CHECKS

Disk Payroll File

Control Totals

Details

To B

To C

File D
PAY 08 INQUIRY

Disk Payroll File

Select Desired Clock Number Card

Clock Number

Last Week's Payroll Register

Weekly Time Sheets

Balance to Totals; If Correct, Go to E

Determine Change Required

Use PAY 16 & PAY 03 to Change the Disk Payroll Record

Does this correct original error? If not, Go to E

Return to Print Where Error Occurred

Only when entire original error has been corrected
PAYROLL RECORD LAYOUTS

Card Forms and Console Keyboard Input

PAY01
Plant no. — 1 digit — keyboard
Week no. of month — 1 digit — keyboard
Check no. — 2 digits — keyboard
Name — 18 blanks — keyboard
Plant name — 32 characters maximum — keyboard
Figure 2 — card

PAY02
Plant no. — 1 digit — keyboard
Figure 3 — card

PAY03
Plant no. — 1 digit — keyboard
Figure 1 — card
Social Security Number, if changed — keyboard
Figure 4 — card
Figure 5 — card

PAY04
Figure 6 — card
Check no. — 5 digits — keyboard
Week no. of month — 1 digit — keyboard
Maximum check amount allowed — 5 digits — keyboard
Figure 7 — card

PAY05
Figure 6 — card
Check no. — 5 digits — keyboard
Check maximum amount — 5 digits — keyboard
Clock no. (if requested) — 4 digits — keyboard

PAY06
Figure 6 — card

PAY07
Plant no. — 1 digit — keyboard

PAY08
Figure 9 — card
Figure 10 — card
Figure 5 — card

PAY09
Figure 11 — card
Figure 12 — card
Figure 13 — card
Figure 14 — card
Figure 15 — card

PAY10
Figure 9 — card
Figure 5 — card
PAY11
   Figure 6 — card
   Figure 8 — card
   Figure 5 — card
   If requested:
      Insurance deduction — 4 digits — keyboard
      Stock deduction — 4 digits — keyboard
      Charity deduction — 4 digits — keyboard
      Miscellaneous deduction — 4 digits — keyboard
PAY12
   Plant no. — 1 digit — keyboard
PAY13
   Plant no. — 1 digit — keyboard
   Individual amount for a plant — 4 digits — keyboard
PAY14
   Plant no. — 1 digit — keyboard
PAY15
   Plant no. — 1 digit — keyboard
PAY16
   Figure 6 — card
   Figure 7 — card

Console Printer and Line Printer Forms for Output

PAY01 — None
PAY02 — None
PAY03 — None
PAY04 — Figure 17
   Figure 17
   Figure 8
PAY05 — Figure 18
PAY06 — Figure 19
PAY07 — Figure 20
PAY08 — Figure 16
PAY09 — Figure 21
PAY10 — Figure 22
PAY11 — Figure 17
PAY12 — None
PAY13 — Figure 23
PAY14 — Figure 24
PAY15 — Figure 25
PAY16 — Figure 26

Disk Record Formats

Employee File — Figure 27
Index to Employee File — Figure 28
Company Record in the Corporation File — Figure 29
Figure 1.

<table>
<thead>
<tr>
<th>Check No.</th>
<th>Pay Rate</th>
<th>Social Security No.</th>
<th>Gross Earnings YTD</th>
<th>FICA YTD</th>
<th>FIT YTD</th>
<th>Local Tax YTD</th>
<th>Credits</th>
<th>Deductions</th>
<th>Credit</th>
<th>Payroll</th>
<th>Union Dues</th>
<th>Blank</th>
<th>Metro St.</th>
<th>Blank</th>
<th>Last Cal.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>1 2 3 4</td>
<td>5 6 7 8 9 10 11 12</td>
<td>13 14 15 16 17 18</td>
<td>19 20 21 22</td>
<td>23 24 25 26 27</td>
<td>28 29 30 31 32</td>
<td>33 34 35 36 37</td>
<td>38 39 40 41 42</td>
<td>43 44 45 46 47</td>
<td>48 49 50 51 52</td>
<td>53 54 55 56 57</td>
<td>58 59 60 61 62</td>
<td>63 64 65 66 67</td>
<td>68 69 70 71 72</td>
<td>73 74 75 76 77 78 79 80 81 82</td>
</tr>
</tbody>
</table>

Figure 2.
### Figure 7

<table>
<thead>
<tr>
<th>Clock No.</th>
<th>Regular Hours</th>
<th>Overtime Hours</th>
<th>Bonus Hours</th>
<th>Special Earnings</th>
<th>Special Earnings</th>
<th>Pay Rate</th>
<th>Gross</th>
<th>Net</th>
<th>FIT</th>
<th>FICA</th>
<th>Local Tax</th>
<th>Credit Union</th>
<th>Total All Other Deductions</th>
<th>Taxable</th>
<th>Total Payroll</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Figure 8

<table>
<thead>
<tr>
<th>Clock No.</th>
<th>Regular Hours</th>
<th>Overtime Hours</th>
<th>Bonus Hours</th>
<th>Special Earnings</th>
<th>Special Earnings</th>
<th>Pay Rate</th>
<th>Gross</th>
<th>Net</th>
<th>FIT</th>
<th>FICA</th>
<th>Local Tax</th>
<th>Credit Union</th>
<th>Total All Other Deductions</th>
<th>Taxable</th>
<th>Total Payroll</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Figure 9,

<table>
<thead>
<tr>
<th>Plant</th>
<th>No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
</tr>
<tr>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>36</td>
<td>37</td>
</tr>
<tr>
<td>38</td>
<td>39</td>
</tr>
<tr>
<td>40</td>
<td>41</td>
</tr>
<tr>
<td>42</td>
<td>43</td>
</tr>
<tr>
<td>44</td>
<td>45</td>
</tr>
<tr>
<td>46</td>
<td>47</td>
</tr>
<tr>
<td>48</td>
<td>49</td>
</tr>
<tr>
<td>50</td>
<td>51</td>
</tr>
<tr>
<td>52</td>
<td>53</td>
</tr>
<tr>
<td>54</td>
<td>55</td>
</tr>
<tr>
<td>56</td>
<td>57</td>
</tr>
<tr>
<td>58</td>
<td>59</td>
</tr>
<tr>
<td>60</td>
<td>61</td>
</tr>
<tr>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>64</td>
<td>65</td>
</tr>
<tr>
<td>66</td>
<td>67</td>
</tr>
<tr>
<td>68</td>
<td>69</td>
</tr>
<tr>
<td>70</td>
<td>71</td>
</tr>
<tr>
<td>72</td>
<td>73</td>
</tr>
<tr>
<td>74</td>
<td>75</td>
</tr>
<tr>
<td>76</td>
<td>77</td>
</tr>
<tr>
<td>78</td>
<td>79</td>
</tr>
<tr>
<td>80</td>
<td>81</td>
</tr>
</tbody>
</table>

Figure 10,

<table>
<thead>
<tr>
<th>Clock No.</th>
<th>Blank</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
</tr>
<tr>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>36</td>
<td>37</td>
</tr>
<tr>
<td>38</td>
<td>39</td>
</tr>
<tr>
<td>40</td>
<td>41</td>
</tr>
<tr>
<td>42</td>
<td>43</td>
</tr>
<tr>
<td>44</td>
<td>45</td>
</tr>
<tr>
<td>46</td>
<td>47</td>
</tr>
<tr>
<td>48</td>
<td>49</td>
</tr>
<tr>
<td>50</td>
<td>51</td>
</tr>
<tr>
<td>52</td>
<td>53</td>
</tr>
<tr>
<td>54</td>
<td>55</td>
</tr>
<tr>
<td>56</td>
<td>57</td>
</tr>
<tr>
<td>58</td>
<td>59</td>
</tr>
<tr>
<td>60</td>
<td>61</td>
</tr>
<tr>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>64</td>
<td>65</td>
</tr>
<tr>
<td>66</td>
<td>67</td>
</tr>
<tr>
<td>68</td>
<td>69</td>
</tr>
<tr>
<td>70</td>
<td>71</td>
</tr>
<tr>
<td>72</td>
<td>73</td>
</tr>
<tr>
<td>74</td>
<td>75</td>
</tr>
<tr>
<td>76</td>
<td>77</td>
</tr>
<tr>
<td>78</td>
<td>79</td>
</tr>
<tr>
<td>80</td>
<td>81</td>
</tr>
</tbody>
</table>

17
Figure 13.

Figure 14.
Figure 15.
<table>
<thead>
<tr>
<th>LINE DESCRIPTION</th>
<th>FIELD HEADINGS/WORD MARKS</th>
<th>8 Lines Per Inch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>IBM 407, 408, 409, 1403, 1404, 1443, and 2203</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Print Span 1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>GLUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>------</td>
</tr>
</tbody>
</table>

**Figure 17. (Cont)**
<table>
<thead>
<tr>
<th>LINE DESCRIPTION</th>
<th>PRINT SPACING</th>
<th>IBM 407, 408, 409, 1403, 1404, 1443, and 2203</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLUE</td>
<td>0</td>
<td>IBM 1403 Models 1 &amp; 4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>IBM 407, 408, 409, and 1403 Models 6 and 7</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>IBM 1403 Models 2, 3, 5, N1 and 1404</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>IBM 1403 Models 1, N1, and 2203</td>
</tr>
</tbody>
</table>

Figure 18.
**Figure 19.**

[Diagram showing a printer spacing chart for IBM 407, 408, 409, 1403, 1404, 1443, and 2203 printers. The chart includes various field headers and word marks for different print spans.]
<table>
<thead>
<tr>
<th>LINE DESCRIPTION</th>
<th>FIELD HEADINGS/WORD MARKS</th>
<th>8 Lines Per Inch</th>
<th>IBM 407, 408, 409, 1403, 1404, 1443, and 2203</th>
</tr>
</thead>
<tbody>
<tr>
<td>On Console Printer</td>
<td></td>
<td></td>
<td>IBM 403 Models 1, 2, 3, 4, and 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 403 Models 6, 7, and 1444</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1404 Models 1, 2, 3, 4, and 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 143 Models 1, N1, and 2203</td>
</tr>
</tbody>
</table>

Produced for each Plant

Figure 20,
<table>
<thead>
<tr>
<th>LINE DESCRIPTION</th>
<th>FIELD HEADINGS/WORD MARKS</th>
<th>6 Lines Per Inch</th>
<th>IBM 407, 408, 409, 1403, 1404, 1443, and 2203</th>
<th>Print Span</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1403 Models 1 &amp; 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 407, 408, 409, 1403, 1404, 1443, and 2203</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 1403 Models 2, 3, 5, M1, and 1404</td>
<td></td>
</tr>
</tbody>
</table>

Figure 22.
Figure 23.
Figure 24.
Figure 25.

Figure 26.
EMPLOYEE information record starting at 109 and continuing thru 156 is current.

Figure 27.
Each record is composed of 1 word. The number of records in the file is the number of employees in the plant plus 25%. The last entry is the record number of the last clock number entered.

---

### Year-to-Date Information

<table>
<thead>
<tr>
<th>25 26</th>
<th>30 31</th>
<th>35 36</th>
<th>40 41</th>
<th>45 46</th>
<th>50 51</th>
<th>55 56</th>
<th>60 61</th>
</tr>
</thead>
</table>

### Quarter-to-Date Information

|-------|-----------|----------|-----------|----------|-----------|----------|-----------|----------|-----------|----------|

Previous 13 weeks

Overtime Rate

---

### For Growth of Record

<table>
<thead>
<tr>
<th>FIT</th>
<th>Local Tax</th>
<th>Credit Union</th>
<th>Charity</th>
<th>Union Ded</th>
<th>Stock</th>
<th>Misc.</th>
<th>For Growth of Record</th>
</tr>
</thead>
</table>

| 150 151 | 156 157 | 160 161 | 166 167 |

Figure 28.
This is the plant information record.

Trade Association Information

General Ledger Account Numbers for Posting

Max. Check Amount
Total Weekly Gross
Total Weekly Net
Final Check No.

Available for Expansion

Figure 29.
PAYROLL PROGRAMS

PAY01: PAYROLL FILE CREATE

<table>
<thead>
<tr>
<th>VARIABLES</th>
<th>IBM 1130 COMPUTING SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VARIABLE SUMMARY SHEET</td>
</tr>
<tr>
<td>NAME</td>
<td>MAX. VALUE</td>
</tr>
<tr>
<td>CKMAX</td>
<td>R½ I T</td>
</tr>
<tr>
<td>COMP</td>
<td>A2 / GI.D</td>
</tr>
<tr>
<td>FIBRE</td>
<td>R²/4 O</td>
</tr>
<tr>
<td>I</td>
<td>I I T</td>
</tr>
<tr>
<td>IG</td>
<td>I / N</td>
</tr>
<tr>
<td>ICHCK</td>
<td>I I T</td>
</tr>
<tr>
<td>ICOL</td>
<td>I I T</td>
</tr>
<tr>
<td>IND</td>
<td>I I T</td>
</tr>
<tr>
<td>INDEX</td>
<td>I O T</td>
</tr>
<tr>
<td>INIT</td>
<td>I I O</td>
</tr>
<tr>
<td>IN1</td>
<td>I I T</td>
</tr>
<tr>
<td>IN2</td>
<td>I / N</td>
</tr>
<tr>
<td>IN3</td>
<td>I / N</td>
</tr>
<tr>
<td>IN4</td>
<td>I / N</td>
</tr>
<tr>
<td>IN5</td>
<td>I / N</td>
</tr>
<tr>
<td>IN6</td>
<td>I / N</td>
</tr>
<tr>
<td>IPD</td>
<td>I O O</td>
</tr>
<tr>
<td>ISUPP</td>
<td>I 13 O</td>
</tr>
<tr>
<td>ITO T</td>
<td>I I T</td>
</tr>
<tr>
<td>IWEK</td>
<td>I I T</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWVA</td>
<td>I1N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>K</td>
<td>I1T</td>
<td>0</td>
<td>Ø</td>
<td>Last card test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAST</td>
<td>I1T</td>
<td>X.XX</td>
<td>Ø</td>
<td>Last record number in file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LBO</td>
<td>I1N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LBT</td>
<td>I1N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMC</td>
<td>I1N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LST</td>
<td>I1T</td>
<td>250 250</td>
<td>Last record number in a file</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lyrh1</td>
<td>I1O</td>
<td>Ø Ø</td>
<td>This year’s accumulation of hours worked for vacation pay</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>I1T</td>
<td>-</td>
<td>Used in DO loop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAR</td>
<td>I1I,0</td>
<td>2</td>
<td>1 Marital status (1-single), (2-married)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUNC</td>
<td>I1N</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NADWHI</td>
<td>I1O</td>
<td>Ø Ø</td>
<td>Additional withholding amount</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>I29I,0</td>
<td>-</td>
<td>Dummy area to allocated space for name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCHCK</td>
<td>I1O</td>
<td>Ø Ø</td>
<td>Check number used for this employee</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCU</td>
<td>I1I,0</td>
<td>X.XX</td>
<td>Credit union deduction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCUDDI</td>
<td>I1O</td>
<td>Ø Ø</td>
<td>Monthly Credit union deductions (in dimes)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDOES</td>
<td>I1I,0</td>
<td>X.XX</td>
<td>Union dues deduction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NINS</td>
<td>I1I,0</td>
<td>X.XX</td>
<td>Insurance deduction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMISC</td>
<td>I1O</td>
<td>Ø Ø</td>
<td>Miscellaneous deductions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOLIT</td>
<td>I1T</td>
<td>6</td>
<td>Plant number</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE*</th>
<th>No. of Words</th>
<th>INPUT/TEMP</th>
<th>Output</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRATE</td>
<td>I</td>
<td>10.300</td>
<td></td>
<td></td>
<td>125</td>
<td></td>
<td>Employee pay rate</td>
</tr>
<tr>
<td>NSEX</td>
<td>I</td>
<td>1,0</td>
<td>3</td>
<td></td>
<td>1</td>
<td></td>
<td>Sex-(1-female), (2-male), (3-trucker)</td>
</tr>
<tr>
<td>NSNAN</td>
<td>I</td>
<td>1,0</td>
<td>9</td>
<td>digits</td>
<td></td>
<td></td>
<td>Social Security number</td>
</tr>
<tr>
<td>NSTAS</td>
<td>I</td>
<td>10</td>
<td>5</td>
<td></td>
<td>1</td>
<td></td>
<td>Employee status-(0-union), (1-trucker), (2-non-union, full time), (3-non-union, part-time), (4-terminated)</td>
</tr>
<tr>
<td>NSTCK</td>
<td>I</td>
<td>1,0</td>
<td>XX.XX</td>
<td></td>
<td>0</td>
<td></td>
<td>Stock deduction</td>
</tr>
<tr>
<td>NSTKD</td>
<td>I</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>Monthly stock deductions</td>
</tr>
<tr>
<td>NUA</td>
<td>I</td>
<td>1,0</td>
<td>XX.XX</td>
<td></td>
<td></td>
<td></td>
<td>United appeal deductions</td>
</tr>
<tr>
<td>NUM</td>
<td>I</td>
<td>1,0</td>
<td>XXXX</td>
<td>1000</td>
<td></td>
<td></td>
<td>Clock number</td>
</tr>
<tr>
<td>NWKMP</td>
<td>I</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>Number of weeks employed</td>
</tr>
<tr>
<td>NWKPD</td>
<td>I</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>Number of weeks paid</td>
</tr>
<tr>
<td>NXMPF</td>
<td>I</td>
<td>10</td>
<td>17</td>
<td>0</td>
<td></td>
<td></td>
<td>Federal exemptions</td>
</tr>
<tr>
<td>NXMPS</td>
<td>I</td>
<td>10</td>
<td>17</td>
<td>0</td>
<td></td>
<td></td>
<td>State exemptions</td>
</tr>
<tr>
<td>QRTD</td>
<td>R%</td>
<td>0</td>
<td>XXXXXXX</td>
<td>0.00</td>
<td></td>
<td></td>
<td>Quarter-to-date information-(gross,(2) FIT,(3) FICA,(4) loc.tax,(5) FICA wages,(6) sick pay)</td>
</tr>
<tr>
<td>YTD</td>
<td>R%</td>
<td>0</td>
<td>XXXXXXX</td>
<td>0.00</td>
<td></td>
<td></td>
<td>Year-to-date information-(gross,(2) FIT,(3) FICA,(4) FICA wages,(5) sick pay,(6) spec.A,(7) spec.B,(8) loc.tax,(9) reg.hrs,(10) OT hrs,(11) bonus hrs,(12) reg. hrs,(13) OT hrs,(14) bonus hrs)</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
PAY01

KEYBOARD PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01

PAY01
C----- INITIALIZE VARIABLES

CKMAX=25000.
IC=1
ICOL=1
INIT=0
INIT=1
IPD=0

DO 68 I=1,13
   68 ISUPP(I)=0
   ITOT(I1)=111
   ITOT(2)=620
   ITOT(3)=620
   ITOT(4)=625
   ITOT(5)=626
   ITOT(6)=627
   ITOT(7)=628
   ITOT(8)=628
   ITOT(9)=0
   ITOT(10)=635
   LHR=0
   NADWH=0
   NHCK=0
   NCH=0
   NSTK=0
   NKMP=0
   NKP=0
   ORTD(5)=0.
   ORTD(6)=0.
   DO 69 M=1,14
      69 YTD(M)=0.
C----- READ PLANT NUMBER, WEEK NUMBER, AND CHECK NUMBER

C----- READ(6,4) NOPLT
   READ(6,4) IWEEK
   READ(6,4) ICHCK
4 FORMAT(I1)
5 FORMAT(I2)

C----- CALCULATE THE FILE NUMBER OF THE INDEX FOR THE CURRENT PLANT.
C----- FINISH INITIALIZING VARIABLES - ITOT(4), ITOT(10), LST

IND=100 + NOPLT
GO TO (C152*53*54*55*56)+NOPLT
51 LST=250
GO TO 57
PAY01 PROGRAM

52 LST=90
   ITOT(10)=0
   GO TO 98
53 LST=200
   ITOT(10)=1723
58 ITOT(4)=621
   GO TO 60
54 LST=50
   GO TO 57
55 LST=150
   ITOT(4)=0
   GO TO 59
56 LST=30
57 ITOT(4)=622
59 ITOT(10)=0

C----
C---- SETUP THE NAME FIELD AND RETRIEVE THE COMPANY NAME.
C----
60 READ(16*1) NAME
3 FORMAT(16A2)
   READ(6*1) COMP
1 FORMAT(16A2)

C---- READ ALL INFORMATION FOR ONE EMPLOYEE AND CHECK FOR LAST CARD.
C----
500 READ(2*1) NUM, NRATE, NSEX, NSSAN, NxMF, YTD(1), YTD(2), YTD(3), YTD(8), NCU, NINS, NSTCK, NUA, NDUES, MAR, K
1 FORMAT(1X,14+13,11+13+12,14,1X+12+7,0+3F5.0+15,214+13+14+6X+12,
8X+11)

C----
C---- IS THIS THE LAST CARD
C----
C---- YES - GO TO 600
C----
C---- NO - GO TO 10
C----
C---- IF(K=9) 10+600+10
C----
C----
C---- SETUP EMPLOYEE STATUS CODE, STATE EXEMPTIONS, AND Q=T=D INFORMATION.
C----
C----
10 NSTAS=1
   NXMPF=NXMF
   QRTD(1)=YTD(1)
   QRTD(2)=YTD(2)
   QRTD(3)=YTD(3)
   QRTD(4)=YTD(8)
C----
C----
PAY01 PROGRAM

C------- EDIT MARITAL STATUS, UNION DUES DEDUCTION, SEX CODE, AND IF
C------- NECESSARY, MODIFY EMPLOYEE STATUS CODE.

C----- IF(MAR) IF(MAR) IF(MAR) IF(MAR) IF(MAR) IF(MAR) IF(MAR)
100 IF(MAR=1) IF(MAR=2) IF(MAR=1) IF(MAR=2) IF(MAR=1) IF(MAR=2) IF(MAR=1)
101 MAR=1
CALL STACK

C----- IF(NDUES) IF(NDUES) IF(NDUES) IF(NDUES) IF(NDUES) IF(NDUES) IF(NDUES)
102 IF(NDUES=0) CALL STACK
103 NDUES=0

C----- CALL STACK
104 NDUAS=3
105 IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3)
106 IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3)
107 IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3) IF(NDUAS=3)
108 NSTAS=2

C----- CALL STACK
109 NSEX=2
GO TO 110

C----- CALL STACK
110 INDEX(ICOL)=NUM

C----- WRITE TO THE DISK.
WRITE(NOPLT'ICOL) NUM, NAME, SSN, NSTAS, NDUES, NKMP, NKPD, MAR, NXM, NCU, NCUDD, NCHCK, NADW, NSTCK, NINS, NMISC, NUA, NSTK, ISUPP, INIT, IPD

C----- GO BACK FOR ANOTHER EMPLOYEE'S INFORMATION
GO TO 500

C----- LAST CARD HAS BEEN READ.

C----- INITIALIZE THE TRADE ASSOCIATION INFORMATION.

C----- 600 DO 650 I=1,8
650 FIBRE(I)=0.

C----- WRITE THE INDEX OF EMPLOYEES FOR THIS PLANT TO DISK.
PAY01 PROGRAM

LAST+ICOL-1 WRITE(IND+1) (INDEX(I)+1),LAST PAY01
PAY01

PAY01 IN THE PLANT TO THE INDEX AND STOP. PAY01
PAY01

PAY01 WRITE(255),NOPLT, COMP, INCHCK, INEWK, FIBRE, ITOT, CKMAX PAY01
PAY01

PAY01 CALL EXIT END PAY01
PAY01

PAY01 VARIABLE ALLOCATIONS
PAY01

PAY01 ICOL, IWVA, MUNC, LBO, LBT, LMC, IN1, IN2, IN3, IN4, IN5, NAME
PAY01

PAY01 NSNAN, IC, I01E, I018, I019, I01A, I01B, I01C, NAME
PAY01

PAY01 NSNAN, IC, I01E, I018, I019, I01A, I01B, I01C, NAME
PAY01

PAY01 IN3, IN4, IN5, IN6, IN7, IN8, IN9, INS
PAY01

PAY01 situation allocations
PAY01

PAY01 CALL SUBPROGRAM
PAY01

PAY01 REAL CONSTANTS
PAY01

PAY01 INTEGER CONSTANTS
PAY01

PAY01 COMMON 0 VARIABLES END OF COMPILATION
// JOB 
// XED PAY01 3

*FILES(1=COLFP) (2=WVAFP) (3=MNCFP) (4=LBQFP) (5=LBTFP) (6=LMCFP),
*FILES(125=PINFO),
*FILE(101=INDEX1) (102=INDEX2) (103=INDEX3) (104=INDEX4) (105=INDEX5) (106=INDEX6)

10012142013323060 02
10022613083284339 02
10032147212982119 01
1004261303224378 02
10052722064538734 02
1016280154432308 01
11072613213710014 02
1218214782927112 01
13471741194511234 01
16033722822445678 02

Listing of input cards

1
1
01
THE CONTAINER CORP.
THE CONTAINER CORP.
THE CONTAINER CORP.

Console Printer input and output
\[ // \text{JOB} \]
\[ // \text{XEQ PAY01 3} \]
\[ *\text{FILES(1,COFP)};*(2,WVAFP);*(3,MNCFP);*(4,LBQFP);*(5,LBTFP);*(6,LMCFP); \]
\[ *\text{FILES(25,PIN)}), \]
\[ *\text{FILE5(101,INDX1);*(102,INDX2);*(103,INDX3);*(104,INDX4);*(105,INDX5);*(106,INDX6).} \]

Output on printer
**IBM 1130 MACHINE SETUP SHEET**

<table>
<thead>
<tr>
<th>PROGRAM NAME:</th>
<th>PROGRAM NUMBER:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM DESCRIPTION:</td>
<td>APPROXIMATE RUNNING TIME:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRINTER</th>
<th>TYPE OF PAPER</th>
<th>NO. OF COPIES</th>
<th>CARRIAGE TAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISKS</th>
<th>DRIVE NUMBER:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CARTRIDGE ID:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWITCH SETTINGS</th>
<th>SWITCH</th>
<th>UP</th>
<th>SWITCH</th>
<th>UP</th>
<th>SWITCH</th>
<th>UP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SWITCH</td>
<td>DOWN</td>
<td>SWITCH</td>
<td>DOWN</td>
<td>SWITCH</td>
<td>DOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INPUT CARDS</th>
<th>SOURCE OF INPUT:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISPOSITION OF OUTPUT:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS
IBM 1130 MACHINE SETUP SHEET

PROGRAM NAME: File Create  PROGRAM NUMBER: PAY01

PROGRAM DESCRIPTION: APPROXIMATE RUNNING TIME:

<table>
<thead>
<tr>
<th>TYPE OF PAPER</th>
<th>NO. OF COPIES</th>
<th>CARRIAGE TAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>1</td>
<td>Standard</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRINTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRIVE NUMBER:</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>Payroll</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CARTRIDGE ID:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payroll</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWITCH SETTINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWITCH</td>
</tr>
<tr>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARDS</td>
</tr>
</tbody>
</table>

SOURCE OF INPUT: 1. Card input from a successful PAY16 edit run
2. Disk must be payroll disk with areas for each plant allocated.

DISPOSITION OF OUTPUT: 1. Detail cards are filed in file A.
2. Disk to be used in PAY02, which should be run next.

FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS
PAY02: ADD NAMES TO THE FILE

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>250</td>
<td>1</td>
<td>Used in DO Loop</td>
</tr>
<tr>
<td>ICLK</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>xxxxxxx</td>
<td>1000</td>
<td>Clock Number</td>
</tr>
<tr>
<td>ICOL</td>
<td>I</td>
<td>T</td>
<td>I</td>
<td>250</td>
<td>1</td>
<td>Record number</td>
</tr>
<tr>
<td>IN1</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>250</td>
<td>1</td>
<td>Record number of an individual employee</td>
</tr>
<tr>
<td>INDEX</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>250</td>
<td>1</td>
<td>Index to plant now being processed</td>
</tr>
<tr>
<td>IN1</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>250</td>
<td>1</td>
<td>Record number in indexes to employee files</td>
</tr>
<tr>
<td>IN2</td>
<td>I</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN3</td>
<td>I</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN4</td>
<td>I</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN5</td>
<td>I</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN6</td>
<td>I</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>ISUPP</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>500</td>
<td>0</td>
<td>Supplemental sick pay</td>
</tr>
<tr>
<td>IWVA</td>
<td>I</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>K</td>
<td>I</td>
<td>T</td>
<td>9</td>
<td></td>
<td></td>
<td>Last card test</td>
</tr>
<tr>
<td>LAST</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>250</td>
<td>0</td>
<td>Last record number in file</td>
</tr>
<tr>
<td>LBO</td>
<td>I</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LBT</td>
<td>I</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LMC</td>
<td>I</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to ICOL</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>Input/Output</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>LST</td>
<td>I</td>
<td>T</td>
<td>250</td>
<td>3</td>
<td></td>
<td>Last record number in a file</td>
</tr>
<tr>
<td>LYRHR</td>
<td>I</td>
<td>T</td>
<td>3333</td>
<td>0</td>
<td></td>
<td>This year's accumulation of hours worked for vacation pay</td>
</tr>
<tr>
<td>MAR</td>
<td>I</td>
<td>T</td>
<td>2</td>
<td>1</td>
<td></td>
<td>Marital status - (1- Single), (2- Married)</td>
</tr>
<tr>
<td>MUNC</td>
<td>I</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td></td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>NADWH</td>
<td>I</td>
<td>T</td>
<td>XXXX</td>
<td>0</td>
<td></td>
<td>Additional withholding amount</td>
</tr>
<tr>
<td>NAME</td>
<td>A2</td>
<td>T</td>
<td>-</td>
<td>-</td>
<td></td>
<td>Dummy area to allocated space for name</td>
</tr>
<tr>
<td>NCHECK</td>
<td>I</td>
<td>T</td>
<td>XXXX</td>
<td>0</td>
<td></td>
<td>Checked number used for this employee</td>
</tr>
<tr>
<td>NCU</td>
<td>I</td>
<td>T</td>
<td>XXXX</td>
<td>0</td>
<td></td>
<td>Credit union deduction amount</td>
</tr>
<tr>
<td>NCUDD</td>
<td>I</td>
<td>T</td>
<td>XXXX</td>
<td>0</td>
<td></td>
<td>Monthly credit union deductions (dollars)</td>
</tr>
<tr>
<td>NDUES</td>
<td>I</td>
<td>T</td>
<td>XXXX</td>
<td>0</td>
<td></td>
<td>Union dues deduction amount</td>
</tr>
<tr>
<td>NINS</td>
<td>I</td>
<td>T</td>
<td>XXXX</td>
<td>0</td>
<td></td>
<td>Insurance deduction amount</td>
</tr>
<tr>
<td>NMISC</td>
<td>I</td>
<td>T</td>
<td>XXXX</td>
<td>0</td>
<td></td>
<td>Miscellaneous deductions amount</td>
</tr>
<tr>
<td>NOPLT</td>
<td>I</td>
<td>I</td>
<td>6</td>
<td>1</td>
<td></td>
<td>Plant number</td>
</tr>
<tr>
<td>NRATE</td>
<td>I</td>
<td>T</td>
<td>3333</td>
<td>125</td>
<td></td>
<td>Employee pay rate</td>
</tr>
<tr>
<td>NSEX</td>
<td>I</td>
<td>T</td>
<td>3</td>
<td>1</td>
<td></td>
<td>Sex-(2-Female), (2-Male), (3-Trucker)</td>
</tr>
<tr>
<td>NSSAN</td>
<td>I</td>
<td>B</td>
<td>9 digits</td>
<td></td>
<td></td>
<td>Social Security number</td>
</tr>
<tr>
<td>NSTAS</td>
<td>I</td>
<td>T</td>
<td>5</td>
<td>1</td>
<td></td>
<td>Employee status - (1- Union), (2-Trucker) (3-Non-union, full time), (4-Non-union, part time), (5-terminated)</td>
</tr>
<tr>
<td>NSTCK</td>
<td>I</td>
<td>T</td>
<td>XXXX</td>
<td>0</td>
<td></td>
<td>Stock deduction amount</td>
</tr>
<tr>
<td>NSTK'D</td>
<td>I</td>
<td>T</td>
<td>XXXX</td>
<td>0</td>
<td></td>
<td>Monthly stock deductions</td>
</tr>
<tr>
<td>NUA</td>
<td>I</td>
<td>T</td>
<td>XXXX</td>
<td>0</td>
<td></td>
<td>United appeal deduction amount</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM</td>
<td>I</td>
<td>1.0</td>
<td>I</td>
<td>1000</td>
<td>0</td>
<td>Clock number in disk record</td>
</tr>
<tr>
<td>NUMB</td>
<td>A, 9</td>
<td>T</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Employee name from card</td>
</tr>
<tr>
<td>NWKMP</td>
<td>I, T</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Number of weeks employed</td>
</tr>
<tr>
<td>NWKPD</td>
<td>I, T</td>
<td>5φ</td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
<td>Number of weeks paid</td>
</tr>
<tr>
<td>NMPF</td>
<td>I, T</td>
<td>17</td>
<td>φ</td>
<td>φ</td>
<td>State exemptions</td>
<td></td>
</tr>
<tr>
<td>NMPF</td>
<td>I, T</td>
<td>17</td>
<td>φ</td>
<td>φ</td>
<td>State exemptions</td>
<td></td>
</tr>
<tr>
<td>QRTD</td>
<td>R, 1B</td>
<td>T</td>
<td>XXXXX</td>
<td>φφφφ</td>
<td>φφφφ</td>
<td>Quarter to date information: (1) gross, (2) FICA, (4) FICA wages, (5) sick pay, (6) spec. A, (7) spec. B, (8) loc tax, (9) reg hours</td>
</tr>
<tr>
<td>YTD</td>
<td>R, 12</td>
<td>T</td>
<td>XXXXX</td>
<td>φφφφ</td>
<td>φφφφ</td>
<td>Year to date information: (1) gross, (2) FICA, (3) FICA wages, (4) sick pay, (5) spec. A, (6) spec. B, (7) loc tax, (8) reg hours, (10) OT hours, (11) bonus hours, (12) reg erns, (13) OT erns, (14) bonus erns</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
Start

Initialize Variables

Read Plant No.

Read Employee Index for This Plant

Read Employee Clock No. and Name from Card

Last Card?

YES
Stop

NO

Locate Employee Record No. in Index

Read Employee Record from Disk

Insert Name in Employee Record

Write Employee Record Back to Disk
// FOR PAY02
* 1053(CARD,TYPETHREE,KEYBOARD, DISK)
** PAY02 PROGRAM
* NAME PAY02
* FONE WORD INTEGERS
* EXTENDED PRECISION
* LIST ALL
C----- JOB NAME = PAYROLL SYSTEM - ADD NAMES TO FILE
C----- JOB NUMBER = PAY02
C----- PROGRAMMER = C.R. KLICK
C----- DATE CODED = 12/30/67
C----- DATE UPDATED =
C-----
C----- FILE  FILE RECORD NO. OF RECORDS PAY02
C----- NAME  NUMBER LENGTH RECORDS PER SECTOR PAY02
C----- INPUT FILES -- 1. INDX1 101 1 250 320 PAY02
C----- 2. INDX2 102 1 90 320 PAY02
C----- 3. INDX3 103 1 200 320 PAY02
C----- 4. INDX4 104 1 50 320 PAY02
C----- 5. INDX5 105 1 150 320 PAY02
C----- 6. INDX6 106 1 30 320 PAY02
C----- 7. COLFP  1 160 250 2  PAY02
C----- 8. WVAFP  2 160 90 2  PAY02
C----- 9. MNCFP  3 160 200 2  PAY02
C----- 10. LBOFP  4 160 50 2  PAY02
C----- 11. LBTFP  5 160 150 2  PAY02
C----- 12. LMCFP  6 160 30 2  PAY02
C----- OUTPUT FILES -- 1. COLFP  1 160 250 2  PAY02
C----- 2. WVAFP  2 160 90 2  PAY02
C----- 3. MNCFP  3 160 200 2  PAY02
C----- 4. LBOFP  4 160 50 2  PAY02
C----- 5. LBTFP  5 160 150 2  PAY02
C----- 6. LMCFP  6 160 30 2  PAY02
C----- ALLOCATE ARRAY STORAGE
C-----
C----- DIMENSION INDEX(250), ISUPP(13), NAME(9), NSSAN(3), NUMB(9),
C-----  ORTD(6), YTD(14)

C----- DEFINE THE FILES FOR THIS PROGRAM AS DESCRIBED ABOVE, AND
C----- EQUIVALENCE THE VARIABLES FOR THE NEXT RECORD NUMBER.
C-----
C----- DEFINE FILE
C----- 1(250,160,U,ICOL), 2(200,160,U,IWVA),
C----- 3(50,160,U,MUNC), 4(50,160,U,LBO),
C----- 5(150,160,U,LBT), 6(30,160,U,LMC) 101(250+1,U,IN1),
C----- 102(90+1,U,IN2), 103(200+1,U,IN3), 104(50+1,U,IN4),
C----- 105(150+1,U,IN5), 106(30+1,U,IN6)
C----- EQUIVALENCE (ICOL,IWVA,MUNC,LBO,LBT,LMC),
PAY02 PROGRAM

C---- SEARCH INDEX FOR EMPLOYEE NUMBER
  • C---
  120 DO 125 I=1, LA5.
     IF(INDEX(I) = ICLK) 125+130+125
  • 125 CONTINUE
  • C---
  C---- IF THE PROGRAM COMES THRU HERE, THE CLOCK NO. IS NOT IN THE INDEX
  • C---
  WRITE(1,4) ICLK
  4 FORMAT('CLOCK NO '14' NOT IN FILE')
  • GO TO 100
  • C---
  C---- READ EMPLOYEE RECORD FROM DISK AND VALIDATE CLOCK NUMBERS
  C----
  130 IND=I
     READ(NOPLT'IND) NUM, NAME, NSSAN, NSTAS, NDUES, NWKMP, NWKPD, MAR, PAY02
  1    XMPP, XMPS, NSEX, NRATE, YTD, QRTD, LyrHR, NCU, PAY02
  2    NCUDD, NCHCK, NADWH, NSTCK, NINS, NMISC, NUA, PAY02
  3    NSTKD, ISUPP, INIT
  • C---
  C---- VALIDATE
  • C--- MATCH — 140
  • C--- NO MATCH — 135
  • C---
  135 WRITE(1*5) NUM, ICLK
     5 FORMAT('CLOCK NO '14' IN FILE DOES NOT AGREE WITH CLOCK NUMBER '14'
  1    ' IN CARD')
  • GO TO 100
  • C---
  C---- UPDATE THE EMPLOYEE NAME FIELD, WRITE HIS RECORD BACK TO THE DISK
  C---- AND THEN GO BACK TO THE READ STATEMENT TO GET THE NAME OF THE
  • C--- NEXT EMPLOYEE.
  • C---
  140 WRITE(NOPLT'IND) NUM, NUMB, NSSAN, NSTAS, NDUES, NWKMP, NWKPD,
  1    MAR, XMPP, XMPS, NSEX, NRATE, YTD, QRTD, LyrHR, PAY02
  2    NCU, NCUDD, NCHCK, NADWH, NSTCK, NINS, NMISC, NUA, PAY02
  3    NUA, NSTKD, ISUPP, INIT
  • C---
  C---- GO BACK FOR ANOTHER EMPLOYEE'S NAME
  • C---
  • GO TO 100
  • C---
  • C---- LAST CARD HAS BEEN READ, STOP.
  C----
  99 CALL EXIT
PAY02 PROGRAM

C--------------------------------------------------------PAY02
END

• VARIABLE ALLOCATIONS
• ICOL =0054  IN=0054  NUNC =0054  LBD =0054  LBT =0054  LNC =0054  IN1 =0055  IN2 =0055  IN3 =0055  IN4 =0055
  IN5 =0055  IN6 =0055  ORTD =008F  INDEX=018B  (SUFP=018B  NAME =018A  Ssn=018A  Nump =0180  NOPLT =018A
  INDEX =0184  LST =0180  LAST =0181  ICLOCK =0183  KD =0184  INDEX =0185  NUM =0186  NTASKS =0187  NUSES =0188
  NWM=0189  NWM=0183  MAX =0188  NMPP=0183  NMP=0180  MSEK =018E  NRATE =018F  LYNHR =0180  NCMD =018C
  NCSR=018C  NCSR=018C  NCSR=018C  NCSR=018C  NCSR=018C  NCSR=018C

• STATEMENT ALLOCATIONS
  1 =0107  2 =0109  4 =010F  5 =0113  6 =0116  90 =0266  100 =0281  120 =0291  125 =02A0  150 =02B0  155 =02C0
  160 =0300  99 =033F

• FEATURES SUPPORTED
• ONE WORD INTEGERS
• EXTENDED PRECISION
• CALLED SUBPROGRAMS
  ELD  ESTD  TYPEZ  SRED  SWRT  SCOMP  SFIO  SIOAI  SIOI  SUBSC  CAROZ  SDFIO  GORED  SDWRT  SDCOM
  601  602  603  604  605  606  607  608  609  610  611  612  613  614  615  616

• INTEGER CONSTANTS
  1=010C  6.010C  100=010C  250=010C  90=010D  200=010D  90=010D  150=010D  30=010D  2=010D

• CORE REQUIREMENTS FOR PAY02
• COMMON 0 VARIABLES 460 PROGRAM 372

END OF COMPILATION
Printer output

```
// JOB
// XEG PAYO2 2
*FILES(1, COLFP) x (2, WVAFP) x (3, MNCFP) x (4, LBOFP) x (5, LBTFP) x (6, LMCFP) *
*FILES(101, INDX1) x (102, INDX2) x (103, INDX3) x (104, INDX4) x (105, INDX5) x (106, INDX6) *

<table>
<thead>
<tr>
<th>Clock No</th>
<th>Name</th>
<th>Rate 1</th>
<th>Rate 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>013 32 3060</td>
<td>ROBT B BADEN</td>
<td>1831.01</td>
<td>1831.01</td>
</tr>
<tr>
<td>083 28 4399</td>
<td>JOHN A HORN</td>
<td>2202.84</td>
<td>2202.84</td>
</tr>
<tr>
<td>712 98 2119</td>
<td>ROBT L SHORES</td>
<td>1906.65</td>
<td>1906.65</td>
</tr>
<tr>
<td>032 24 4378</td>
<td>JOHN W CUSSEN</td>
<td>2286.25</td>
<td>2286.25</td>
</tr>
<tr>
<td>814 63 8734</td>
<td>JOSEPH MONTANO</td>
<td>3176.73</td>
<td>3176.73</td>
</tr>
<tr>
<td>941 03 2308</td>
<td>DONALD MILLER</td>
<td>1342.00</td>
<td>1342.00</td>
</tr>
<tr>
<td>213 71 0014</td>
<td>A E TAYLOR</td>
<td>2233.03</td>
<td>2233.03</td>
</tr>
<tr>
<td>782 92 7112</td>
<td>DAVID A HUBBARD</td>
<td>1923.58</td>
<td>1923.58</td>
</tr>
<tr>
<td>194 51 1234</td>
<td>FRANK T Dolen</td>
<td>1475.89</td>
<td>1475.89</td>
</tr>
<tr>
<td>822 44 5678</td>
<td>AL REYNOLDS</td>
<td>3142.25</td>
<td>3142.25</td>
</tr>
</tbody>
</table>
```

Input cards

```
// JOB
// XEG PAYO2 2
*FILES(1, COLFP) x (2, WVAFP) x (3, MNCFP) x (4, LBOFP) x (5, LBTFP) x (6, LMCFP) *
*FILES(101, INDX1) x (102, INDX2) x (103, INDX3) x (104, INDX4) x (105, INDX5) x (106, INDX6) *
1001 ROBT B BADEN
1002 JOHN A HORN
1003 ROBT L SHORES
1004 JOHN W CUSSEN
1005 JOSEPH MONTANO
1009 THIS IS A MISTAKE
1016 DONALD MILLER
1107 A E TAYLOR
1218 DAVID A HUBBARD
1347 FRANK T Dolen
1603 AL REYNOLDS
```

Console Printer output

```
1  CLOCK NO 1009 NOT IN FILE
1  CLOCK NO 1017 NOT IN FILE
```
**IBM 1130 MACHINE SETUP SHEET**

<table>
<thead>
<tr>
<th>PROGRAM NAME:</th>
<th>Add names to the file</th>
<th>PROGRAM NUMBER:</th>
<th>PAY02</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM DESCRIPTION:</td>
<td></td>
<td>APPROXIMATE RUNNING TIME:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRINTER TYPE</th>
<th>Standard</th>
<th>NO. OF COPIES</th>
<th>Standard</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>DISKS DRIVE NUMBER:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARTRIDGE ID:</td>
<td>Payroll</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWITCH SETTINGS</th>
<th>UP</th>
<th>DOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWITCH UP</td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td>SWITCH DOWN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**INPUT CARDS**

```
NAME & CLOCK
NO. CARDS
"FIXEQ PAY02"
"JOB"
```

**SOURCE OF INPUT:**
1. Card input for a successful PAY01 edit run.  
2. Disk must be payroll disk from PAY01.

**DISPOSITION OF OUTPUT:**
1. Name and Clock No. cards are filed in file B.  
2. Disk to be used in PAY03, which should be run next.

**FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS**
PAY03: CHANGES TO THE FILE

<table>
<thead>
<tr>
<th>VARIABLES</th>
<th>IBM 1130 COMPUTING SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NAME</strong></td>
<td><strong>MODE</strong></td>
</tr>
<tr>
<td>I</td>
<td>I / T</td>
</tr>
<tr>
<td>ICHNG</td>
<td>I / I</td>
</tr>
<tr>
<td>ICLK</td>
<td>I / I</td>
</tr>
<tr>
<td>ICOL</td>
<td>I / T</td>
</tr>
<tr>
<td>IND</td>
<td>I / T</td>
</tr>
<tr>
<td>INDEX</td>
<td>I / T</td>
</tr>
<tr>
<td>INIT</td>
<td>I / T</td>
</tr>
<tr>
<td>INDX</td>
<td>I / T</td>
</tr>
<tr>
<td>IN1</td>
<td>I / T</td>
</tr>
<tr>
<td>IN2</td>
<td>I / N</td>
</tr>
<tr>
<td>IN3</td>
<td>I / N</td>
</tr>
<tr>
<td>IN4</td>
<td>I / N</td>
</tr>
<tr>
<td>IN5</td>
<td>I / N</td>
</tr>
<tr>
<td>ING</td>
<td>I / N</td>
</tr>
<tr>
<td>IPO</td>
<td>I / T</td>
</tr>
<tr>
<td>ISUPP</td>
<td>I / T</td>
</tr>
<tr>
<td>IWVA</td>
<td>I / N</td>
</tr>
<tr>
<td>K</td>
<td>I / T</td>
</tr>
<tr>
<td>LAST</td>
<td>I / T</td>
</tr>
<tr>
<td>LBO</td>
<td>I / N</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBT</td>
<td>I/I</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LMC</td>
<td>I/I</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LST</td>
<td>I/T</td>
<td>250</td>
<td>3</td>
<td></td>
<td></td>
<td>Maximum number of records in a file</td>
</tr>
<tr>
<td>LYRHR</td>
<td>I/T</td>
<td>3</td>
<td>#</td>
<td></td>
<td>#</td>
<td>This year's accumulation of hours worked for vacation pay</td>
</tr>
<tr>
<td>MAR</td>
<td>I/T</td>
<td>2</td>
<td>1</td>
<td></td>
<td>1</td>
<td>Marital status - (1-single), (2-married)</td>
</tr>
<tr>
<td>MUNC</td>
<td>I/I</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>NADWH</td>
<td>I/T</td>
<td>XXXX</td>
<td>#</td>
<td></td>
<td>#</td>
<td>Additional withholding amount</td>
</tr>
<tr>
<td>NAME</td>
<td>A2</td>
<td>T</td>
<td>-</td>
<td></td>
<td>-</td>
<td>Employee name</td>
</tr>
<tr>
<td>NCHCK</td>
<td>I/T</td>
<td>XXXX</td>
<td>#</td>
<td></td>
<td>#</td>
<td>Check number used for this employee</td>
</tr>
<tr>
<td>NCU</td>
<td>I/T</td>
<td>XXXX</td>
<td>#</td>
<td></td>
<td>#</td>
<td>Credit union deduction amount</td>
</tr>
<tr>
<td>NCUDD</td>
<td>I/T</td>
<td>XXXX</td>
<td>#</td>
<td></td>
<td>#</td>
<td>Monthly credit union deductions (in dimes)</td>
</tr>
<tr>
<td>NDDS</td>
<td>I/T</td>
<td>XXXX</td>
<td>#</td>
<td></td>
<td>#</td>
<td>Union dues deductions amount</td>
</tr>
<tr>
<td>NEW</td>
<td>I/I</td>
<td>XXX,XX</td>
<td>#</td>
<td></td>
<td>#</td>
<td>New information used in change specified by code (ICHNG)</td>
</tr>
<tr>
<td>NINS</td>
<td>I/T</td>
<td>XX,XX</td>
<td>#</td>
<td></td>
<td>#</td>
<td>Insurance deduction amount</td>
</tr>
<tr>
<td>NMISC</td>
<td>I/T</td>
<td>XXXX</td>
<td>#</td>
<td></td>
<td>#</td>
<td>Miscellaneous deductions amount</td>
</tr>
<tr>
<td>NOPLT</td>
<td>I/I</td>
<td>G</td>
<td>1</td>
<td></td>
<td>1</td>
<td>Plant number</td>
</tr>
<tr>
<td>NRATE</td>
<td>I/T</td>
<td>3.25</td>
<td>1.25</td>
<td></td>
<td></td>
<td>Employee pay rate</td>
</tr>
<tr>
<td>NSEX</td>
<td>I/T</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td>Sex - (1-female), (2-male), (3-trucker)</td>
</tr>
<tr>
<td>NSSAN</td>
<td>B/T</td>
<td>Always 3 digits</td>
<td>Social Security number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSTAS</td>
<td>I/T</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td>Employee status, (1-union), (2-trucker), (3-non-union, full time), (4-non-union, part time), (5-terminated)</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
### FUNCTION OF VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSTCK</td>
<td>Stock deduction amount</td>
</tr>
<tr>
<td>NSTKDM</td>
<td>Monthly stock deductions</td>
</tr>
<tr>
<td>NUA</td>
<td>United appeal deduction amount</td>
</tr>
<tr>
<td>NUM</td>
<td>Clock number in disk record</td>
</tr>
<tr>
<td>NUMB</td>
<td>Last three digits of clock number</td>
</tr>
<tr>
<td>NWKMP</td>
<td>Number of weeks employed</td>
</tr>
<tr>
<td>NWKPD</td>
<td>Number of weeks paid</td>
</tr>
<tr>
<td>NxMPF</td>
<td>Federal exemptions</td>
</tr>
<tr>
<td>NxMPS</td>
<td>State exemptions</td>
</tr>
<tr>
<td>QRKT D</td>
<td>Quarter-to-date information: (1) gross, (2) FIT, (3) FICA, (4) loc. tax, (5) FICA wages, (6) sick pay</td>
</tr>
<tr>
<td>YTD</td>
<td>Year-to-date information: (1) gross, (2) FICA, (3) FIT, (4) FICA wages, (5) sick pay, (6) spec. A, (7) spec. B, (8) loc. tax, (9) reg. hours, (10) OT hours, (11) bonus hours, (12) reg. erns, (13) OT erns, (14) bonus erns.</td>
</tr>
</tbody>
</table>

* Mode: I = integer, R = real, D = decimal, A = alphabetic
Write to Disk Employee Index for This Plant

Start

Initialize Variables

Read Plant No.

Read Employee Index for This Plant

Read a Change Card

Last Change Card?

Yes

No

Read Employee Record from Disk

Make Change to Employee Record

Write Employee Record Back to Disk

Validate Change Code

Locate Employee Record No. in Index

Write to Disk Employee Index for This Plant

Read Card

Is This Another Plant?

Yes

No

STOP
* FOR
  * IOCS(CARD,TYPEWRITER,KEYBOARD +DISK)
  * NAME PAY03
  * ONE WORD INTEGERS
  * EXTENDED PRECISION
  * LIST ALL

C---- JOB NAME     -- PAYROLL SYSTEM  -- CHANGES TO THE FILE
C---- JOB NUMBER   -- PAY03
C---- PROGRAMMER   -- C.R.KLICK
C---- DATE CODED   -- 01/06/68
C---- DATE UPDATED --

C---- INPUT FILES
  1. COLFP   160 250 2
  2. WVAFP   160 90 2
  3. MNCFP   160 200 2
  4. LBOPP   160 50 2
  5. LBTOP   160 150 2
  6. LMCFP   160 30 2
  7. INDX1   101 1 250 320
  8. INDX2   102 1 90 320
  9. INDX3   103 1 200 320
  10. INDX4  104 1 90 320
  11. INDX5  105 1 150 320
  12. INDX6  106 1 30 320

C---- OUTPUT FILES
  1. COLFP   160 250 2
  2. WVAFP   160 90 2
  3. MNCFP   160 200 2
  4. LBOPP   160 50 2
  5. LBTOP   160 150 2
  6. LMCFP   160 30 2
  7. INDX1   101 1 250 320
  8. INDX2   102 1 90 320
  9. INDX3   103 1 200 320
 10. INDX4  104 1 90 320
 11. INDX5  105 1 150 320
 12. INDX6  106 1 30 320

C---- ALLOCATE ARRAY STORAGE
  DIMENSION INDEX(250), ISUPP(13), NAME(9), NSSAN(3), QRTD(6),
  YTD(14)

C---- DEFINE THE FILES FOR THIS PROGRAM AS DESCRIBED ABOVE, AND
C---- EQUIVALENCE THE VARIABLES FOR THE NEXT RECORD NUMBER.

DEFINE FILE 1(250,160+ICOL), 2(90,160+IWA)
INITIALIZE VARIABLES

1000 ICOL=1

IN1=1

READ PLANT NUMBER. CALCULATE THE FILE NUMBER OF THE INDEX FOR THE CURRENT PLANT. FINISH INITIALIZING VARIABLES.

READ(2,1) NOPLT

FORMAT(I1)

INDX=100 + NOPLT

GO TO (80,81,82,83,84,85),NOPLT

80 LST=250

GO TO 90

81 LST=90

GO TO 90

82 LST=200

GO TO 90

83 LST=50

GO TO 90

84 LST=150

GO TO 90

85 LST=30

READ THE EMPLOYEE INDEX FOR THIS PLANT, READ A CHANGE CARD, CHECK FOR LAST CHANGE CARD. AND VALIDATE PLANT NUMBERS, CHANGE CODE AND FIND CLOCK NUMBER IN INDEX.

90 READ(INDX*LST) LAST

READ(INDX*1) (INDEX(I=1)+LAST)

100 READ(2*2) ICLOCK, INUMB, ICHNG, NEW, K

1 FORMAT(I13,12X,I5*8A,11)

IS THIS LAST CARD

YES = GO TO 99

NO = GO TO 101
C-----

* IF(K-9) 101,99,101

C-----

* DO PLANT NUMBERS AGREE

C-----

* YES - GO TO 105

C-----

* NO - GO TO 95

C-----

101 IF(NOPLT=ICLCK) 95,105,95

C-----

* IF THE PROGRAM COMES THRU HERE, THE PLANT NUMBERS DO NOT AGREE.

C-----

95 WRITE(1,3) ICLCK, NUMB

3 FORMAT("PLANT NOS DISAGREE FOR CLOCK NUMBER '11,13")

C-----

CALL STACK

GO TO 100

C-----

* PUT PARTS OF CLOCK NUMBER TOGETHER AND CHECK CHANGE CODE

C-----

105 ICLCK=ICLCK * 100 + NUMB

C-----

* CHNG MUST BE BETWEEN 1 AND 16.

C-----

* IF NOT GO TO 104

C-----

* IF O.K., GO TO APPROPRIATE CHANGE ROUTINE.

C-----

IF(CHNG) 104,104,106

106 IF(CHNG=16) 110,120,120

C-----

* CODE 14 INDICATES NEW EMPLOYEE. IF SO, GO TO 500.

C-----

104 IF(CHNG=14) 120,500,120

C-----

* IF THE PROGRAM COMES THRU HERE, THE CHANGE CODE IS INVALID.

C-----

110 WRITE(1,8) ICLCK

8 FORMAT("INVALID CHANGE CODE FOR '14")

C-----

CALL STACK

GO TO 100

C-----

* LOCATE CLOCK NUMBER IN INDEX.

C-----

120 DO 125 I=1,LAST

IF(INDEX(I)=ICLCK) 125,130,125

C-----

* GO TO 125 IF NO MATCH, 130 IF FOUND.

C-----

125 CONTINUE

C-----

* IF THE PROGRAM COMES THRU HERE, THE CLOCK NO. IS NOT IN THE INDEX.

C-----

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03

PAY03
WRITE(1,4) ICLK
4 FORMAT('CLOCK NO '14' NOT IN FILE')
GO TO 100

C----- READ EMPLOYEE RECORD FROM DISK AND VALIDATE CLOCK NUMBERS.
C-----
130 IND=1
READ(NOPLT'LIND) NUM, NAME, NSSAN, NSTAS, NDUES, NWKMP, NWKPD, MAR.
1 NXMPF, NXMPS, NSEX, NRATE, YTD, ORTD, LYNHR, NCU.
2 NCUDO, NCHCK, NADWH, NSTCK, NINS, NMISC, NUA.
3 NSTKD, ISUPP, INIT

C----- VALIDATE
C----- MATCH - 140
C----- NO MATCH - 135
C-----
135 WRITE(1,5) ICLK
5 FORMAT('CLOCK NUMBERS DO NOT AGREE FOR '14')
CALL STACK
GO TO 100

C----- GO TO THE APPROPRIATE CHANGE ROUTINE.
C-----
140 GO TO (141,142,143,144,145,146,147,148,149,150,151,152,153,104,155,156,157,158,159,160)
1 255,156,157,158,159,160,1ICHNG
141 NRATE=NEW
GO TO 550
142 NCU=NEW
GO TO 550
143 NDUES=NEW
GO TO 550
144 NSTAS=NEW
GO TO 550
145 MAR=NEW
GO TO 550
146 NWKMP=NEW
GO TO 550
147 NXMPS=NEW
GO TO 550
148 NSEX=NEW
GO TO 550
149 NADWH=NEW
GO TO 550
150 NSTCK=NEW
  GO TO 550
151 NINS=NEW
  GO TO 550
152 NMISC=NI4
  GO TO 550
153 NUA=NEW
  GO TO 550
155 INIT=NEW
  NSTAS=1
  GO TO 550
156 WRITE(1,11) NUM
  11 FORMAT('ENTER SSAN FOR 'I4)
  INDEX=10) NSSAN
  10 FORMAT('I3ID14A1)
  GO TO 550
  500 READ(2,6) NUM, NAME, NSSAN, NSTAS, MAR, NMPPF, NMXPS, NSEX, NRATE, PAY03
     PAY03
  6 FORMAT('I14O2,13,12,14,511,13,514,13) PAY03
  40
  C	 PAY03
  C	  IS THIS NUMBER, NUM, ALREADY IN INDEX PAY03
  C	  YES - 513 PAY03
  C	  NO - SET UP DISK RECORD PAY03
  C	 DO 504 I=1, LAST PAY03
  504 CONTINUE PAY03
  513 WRITE(1,7) NUM PAY03
     7 FORMAT('CLOCK NUMBER 'I4' IS DUPLICATED') PAY03
  501 YTD(I)=0 PAY03
  502 QRTD(I)=3 PAY03
  503 ISUPP(I)=3 PAY03
  504 LHRHY=0 PAY03
  505 NCDD=0 PAY03
  506 NCHCK=0 PAY03
Section
35

Page

Subsections
10

20

70

PAGE 06

•
•
•

C

C
C
C
C

C

•

•
•
•
•
•

•
•

•
•
•
•
•
•

•
•

41
ID

PAYO3
PAY03
PAYO3
PAY03
PAY03
PAY03
WRITE THE EMPLOYEE RECORD TO THE DISK.
PAY03
GO BACK TO THE READ STATEMENT TO GET ANOTHER CHANGE FOR THIS
PAY03
PLANT.
PAY03
PAY03
550 WRITEINOPLPIND) NUM. NAME, NSSAN. NSTAS. NDUES. NWKMP. NWKPD.
MAR, NXMPF. NXMPS. NSEX, NRATE, YTD. ORTD. LYRHR,PAY03
1
PAY03
NCU, NCUDD. NCHCK, NADWHo NSTCK. NINS. NMISC.
2
PAY03
NSA, NSTKDo ISUPP, INIT. IPD
3
PAY03
PAY03
GO BACK TO READ
PAY03
PAY03
GO TO 1( 7
PAY03
PAY03
PAYO3
WRITE BACK TO DISK THE EMPLOYEE INDEX FOR THIS PLANT.
PAY03
PAYO3
99 WRITEIINDX)LSTI LAST
Mg:
WRITE)IND!)1) .. (INDE!(!):1:1:LAST)
LAST.LAST • 1
IND.LAST
INDEX(LAST).NUM

C

C
C
C
C
C
C
C
C
C
C
C

C
C
C

READ A CARD. IF K IS NOT 9 THERE ARE CHANGES TO ANOTHER PLANT.

PAY03
PAY03
PAY03
PAYO3
P0703
PAY03
P0703
PAY03
PAY03
PAY03
PAY03
PAY03
PAY03

READ(2.9) K
9 FORMAT(79)011)
K NOT 9 MEANS MORE CHANGES. GO TO 1000.
K EQUAL TO 9 MEANS END OF RUN. GO TO 1001.
IFIK - 9) 1000,1001,1000
THIS IS END OF RUN. STOP.
1001 CALL EXIT
END

VARIABLE ALLOCATIONS
ICOL .0054 IWVA =0054
INS =0055 INS .0055
LOT .01A7 LAST .01A8
NSTAS.0181 NDUES.0182
NCU .0188 NCUDD.01BC
IPD .0105

MUNC =0054
ORTD .0065
I
.0309
NWKMP.011:13
NCHCK.01BD

LBO .0054
YTD .008F
ICLCK.01AA
NWKPD=0184
NADWH=01BE

LBT .0054
INDEX=0188
NUMB =01AB
MAR =0185
NSTCK=018F

LMC .0054
ISUPP.0198
ICHNG.01AC
NXMPF.0186
SINS .01C0

151 =0055
NAME .01A1
NEW .01AD
NXMPS.0187
NMISC.01C1

IN2 =0055
NSSAN.01A4
K
=01AE
NSEX .0188
NUA .01C2

STATEMENT ALLOCATIONS
1
.01DB 2
.0100
9
.0259 1000 .0271
=02E4
101 . 020E 95
140 .0386 141 .039A
150 .03CE 151 =03D4
502 •0473 503 =0488

3
80
105
142
152
550

8
81
106
143
153
99

.01FA
=0294
=02FD
=03A6
=03E0
=04F9

4
=0209
82
.029A
110 .0305
144 =035C
155 .03E6
1001 .0521

5
83
104
145
156

11
84
120
146
500

10
85
125
147
513

.01E4
.028E
.02F0
=03A0
•0300
=04E18

=0218
=0200
.0300
=0382
.03F0

.02213
=02A6
=0317
=0388
=03FE

=0236
.02AC
.0326
.038C
.0430

153 =0055
NOPLT.01A5
IND .U1AF
NRATE.0189
NSTI(D.U1C3
=0230
.6280
.U336
-03C2
.043A

6
90
130
140
504

IN4 =0055
INDX .0106
NUM =01E10
LYRHR=01BA
INIT .01C4
7
106
135
149
501

.0247
.62CB
=0370
=03C=
=045E

FEATURES SUPPORTED
ONE WORD INTEGERS
EXTENDED PRECISION
IOCS
CALLED SUBPROGRAMS
STACK
ELD
ESTO
SDAI
SOWRT
SDCOM

ESTOX
SDAF

TYPEZ
SDI%

SRED
SDI

SWRT

SCOMP

SFIO

5I001

5101

SUBSC

CAROL

SDFIO

SORED

REAL CONSTANTS
.000000000E 00..01C8

•

INTEGER CONSTANTS
1.01CB
2.01CC
1000.0105
16.0106

•

CORE REQUIREMENTS FOR PAY03
COMMON
0 VARIABLES

•

PAY03
Mg:

100.01CD
14.0107

250=010E
6.0108

456 PROGRAM

90.01CF
0.0109

858

END OF COMPILATION

66

200.0100
13.01DA

50.0101

150=0102

30.0103

9.0104


// JOB
// KEQ PAY03 2
*FILES(1;COLFP);(2;WVAFP);(3;HCFP);(4;LBQFP);(5;LBTFP);(6;LMCPF);
*FILES(101;INDX1);(102;INDX2);(103;INDX3);(104;INDX4);(105;INDX5);(106;INDX6)
  1
  10010100261
  10040600004
  10160500002
  10170100261

Input cards
<table>
<thead>
<tr>
<th>Section</th>
<th>Subsections</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

Printer output

```
// JOB
// XEQ PAY03 2
*FILES(101+INDX1),(102+INDX2),(103+INDX3),(104+INDX4),(105+INDX5),(106+INDX6)
```
IBM 1130 MACHINE SETUP SHEET

PROGRAM NAME: Changes to the file  PROGRAM NUMBER: PAY03

PROGRAM DESCRIPTION: APPROXIMATE RUNNING TIME:

<table>
<thead>
<tr>
<th>PRINTER</th>
<th>TYPE OF PAPER</th>
<th>NO. OF COPIES</th>
<th>CARRIAGE TAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard</td>
<td>1</td>
<td>Standard</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISKS</th>
<th>DRIVE NUMBER:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CARTRIDGE ID:</td>
<td>Payroll</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWITCH SETTINGS</th>
<th>SWITCH UP</th>
<th>SWITCH DOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWITCH</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

INPUT CARDS

SOURCE OF INPUT:
1. Card input from a successful PAY10 edit run.
2. Disk must be payroll disk from files.

DISPOSITION OF OUTPUT:
1. Change cards are filed in file C.
2. Disk is returned to storage for use with PAY04.

FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS
### IBM 407, 408, 409, 1403, 1404, 1443, and 2203

**PRINTER SPACING CHART**

#### Field Headings/Word Marks

<table>
<thead>
<tr>
<th>LINE DESCRIPTION</th>
<th>FIELD HEADINGS/WORD MARKS</th>
<th>8 Lines Per Inch</th>
</tr>
</thead>
</table>

#### IBM 407, 408, 409, 1403, 1404, 1443, and 2203

**PRINTED REPRINT**

**COMPUTER NAME:** IBM 407, 408, 409, 1403, 1404, 1443, and 2203

**PRINT SPACED:**

- IBM 407, 408, and 1403 Models 1 & 4
- IBM 407, 408, and 1403 Models 2, 3, and 7
- IBM 407, 408, 409, and 1403 Models 6 and 7
- IBM 1403 Models 1, 5, 6, and 7
- IBM 1403 Models 8, 9, and 10

**PRINT SPAN:**

- IBM 407, 408, 409, and 1403 Models 6 and 7
- IBM 1403 Models 1, 5, 6, and 7
- IBM 1403 Models 8, 9, and 10

---

**PAYROLL CALCULATIONS AND PAYROLL REGISTER**

---

**INTERNATIONAL BUSINESS MACHINES CORPORATION**

---

**END OF PAGE**
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE*</th>
<th>No. of Words</th>
<th>INPUT/TEMP.</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Used for zero balance check</td>
</tr>
<tr>
<td>AD</td>
<td>R</td>
<td>3</td>
<td>T</td>
<td>xxx.xx</td>
<td>0.00</td>
<td>Used to calculate overtime rate</td>
</tr>
<tr>
<td>ADREG</td>
<td>R</td>
<td>3</td>
<td>T</td>
<td>xxx.xx</td>
<td>0.00</td>
<td>Used to calculate overtime rate</td>
</tr>
<tr>
<td>ATAX</td>
<td>R</td>
<td>3</td>
<td>T</td>
<td>xxx.xx</td>
<td>0.00</td>
<td>After-tax income</td>
</tr>
<tr>
<td>B</td>
<td>R</td>
<td>3</td>
<td>T</td>
<td>0.00</td>
<td>0.00</td>
<td>Used for zero balance check</td>
</tr>
<tr>
<td>BERN</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>xxx.xx</td>
<td>0.00</td>
<td>Bonus earnings</td>
</tr>
<tr>
<td>BHR</td>
<td>R</td>
<td>3</td>
<td>1.0</td>
<td>xxx.xx</td>
<td>0.00</td>
<td>Bonus, hours</td>
</tr>
<tr>
<td>C</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Used for zero-balance check</td>
</tr>
<tr>
<td>CKMAX</td>
<td>R</td>
<td>3</td>
<td>T</td>
<td>000.00</td>
<td>0.00</td>
<td>Maximum check amount for a file</td>
</tr>
<tr>
<td>CNET</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>xxx.xx</td>
<td>0.00</td>
<td>Net amount of individual check</td>
</tr>
<tr>
<td>COMP</td>
<td>A</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>-</td>
<td>Company name</td>
</tr>
<tr>
<td>D</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Used for zero-balance check</td>
</tr>
<tr>
<td>ERNGS</td>
<td>R</td>
<td>3</td>
<td>T</td>
<td>xxx.xx</td>
<td>0.00</td>
<td>Fica taxable wages</td>
</tr>
<tr>
<td>FIBRE</td>
<td>R</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0.00</td>
<td>Trade association reports</td>
</tr>
<tr>
<td>GROSS</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>xxx.xx</td>
<td>0.00</td>
<td>Gross amount of individual check</td>
</tr>
<tr>
<td>HLDY</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>xxx.xx</td>
<td>0.00</td>
<td>Individual's holiday pay</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>1</td>
<td>T</td>
<td></td>
<td></td>
<td>Used in DO loop</td>
</tr>
<tr>
<td>IC</td>
<td>I</td>
<td>1</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>ICchk</td>
<td>I</td>
<td>1</td>
<td>T</td>
<td>set for</td>
<td>Beginning check number when writing checks</td>
<td></td>
</tr>
<tr>
<td>ICLK</td>
<td>I</td>
<td>1</td>
<td>T</td>
<td>6</td>
<td>1</td>
<td>First digit of clock number</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICNT</td>
<td>I/0</td>
<td>10</td>
<td></td>
<td>0</td>
<td></td>
<td>Sequence number for journal (should correspond to CR #)</td>
</tr>
<tr>
<td>ICOL</td>
<td>I/T</td>
<td>250</td>
<td></td>
<td>1</td>
<td></td>
<td>Record number in employee files, setup by plant</td>
</tr>
<tr>
<td>ICU</td>
<td>I/O</td>
<td>10</td>
<td></td>
<td>0</td>
<td></td>
<td>Individual's credit union deduction</td>
</tr>
<tr>
<td>IDATE</td>
<td>I/T</td>
<td>10</td>
<td></td>
<td>0</td>
<td></td>
<td>Pay date</td>
</tr>
<tr>
<td>IDED</td>
<td>I/O</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>Total of individual's insurance, stock, charity and misc. deductions/paid period</td>
</tr>
<tr>
<td>IFICA</td>
<td>I/O</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>Individual's fica tax</td>
</tr>
<tr>
<td>IFILL</td>
<td>I/T</td>
<td>250</td>
<td></td>
<td>2</td>
<td></td>
<td>Indicates deduction not made</td>
</tr>
<tr>
<td>IINS</td>
<td>I/O</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>Individual's insurance deduction</td>
</tr>
<tr>
<td>IIST</td>
<td>I/T</td>
<td>250</td>
<td></td>
<td>2</td>
<td></td>
<td>Last record number in a file</td>
</tr>
<tr>
<td>IMISC</td>
<td>I/O</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>Individual's misc. deduction</td>
</tr>
<tr>
<td>IND</td>
<td>I/T</td>
<td>250</td>
<td></td>
<td>101</td>
<td></td>
<td>File number of index for a plant. P+100</td>
</tr>
<tr>
<td>INDEX</td>
<td>I/T</td>
<td>250</td>
<td></td>
<td>100</td>
<td></td>
<td>Index to plant now being processed</td>
</tr>
<tr>
<td>INDX</td>
<td>I/T</td>
<td>250</td>
<td></td>
<td>101</td>
<td></td>
<td>Index file number (plant no. +100)</td>
</tr>
<tr>
<td>INIT</td>
<td>I/T</td>
<td>250</td>
<td></td>
<td>0</td>
<td></td>
<td>Union initiation fee</td>
</tr>
<tr>
<td>IN1</td>
<td>I/T</td>
<td>250</td>
<td></td>
<td>0</td>
<td></td>
<td>Record number in indexes to employee files</td>
</tr>
<tr>
<td>IN2</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN3</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN4</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN5</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN6</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
<td>Equivalent to IN1</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOTRT</td>
<td>I IT</td>
<td>500</td>
<td>φ</td>
<td>Overtime pay rate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPAGE</td>
<td>I IO</td>
<td>20</td>
<td>1</td>
<td>Page number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPD</td>
<td>I IO</td>
<td>φ</td>
<td>φ</td>
<td>Indicates status of record in processing cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISTCK</td>
<td>I IO</td>
<td>2φφφ</td>
<td>φ</td>
<td>Individual's stock deduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISUPP</td>
<td>I 130</td>
<td>XXX,xx</td>
<td>φ</td>
<td>Supplemental sick pay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITOT</td>
<td>I IT</td>
<td>1723</td>
<td>φ</td>
<td>Account number for posting to in general ledger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IUA</td>
<td>I IO</td>
<td>3φφφ</td>
<td>φ</td>
<td>Individual's charity deduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IUD</td>
<td>I IO</td>
<td>15φφφ</td>
<td>φ</td>
<td>Individual's union dues deduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IVRAT</td>
<td>I IO</td>
<td>500</td>
<td>φ</td>
<td>Average pay rate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IWEK</td>
<td>I IT</td>
<td>5</td>
<td>1</td>
<td>Week of month</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IWVA</td>
<td>I IN</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>I IT</td>
<td>9</td>
<td>φ</td>
<td>Last-card test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KARD</td>
<td>I I 1</td>
<td>9</td>
<td>φ</td>
<td>C.C. 8φ for last-card test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KO</td>
<td>A IO</td>
<td>5</td>
<td>φ</td>
<td>Special earnings code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KODE</td>
<td>I 3/1</td>
<td>I 9</td>
<td>φ</td>
<td>Special earnings code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KPLNT</td>
<td>I I I</td>
<td>6</td>
<td>φ</td>
<td>Plant number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAST</td>
<td>I IT</td>
<td>XXX</td>
<td>φ</td>
<td>Last record number in file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LBO</td>
<td>I IN</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LBT</td>
<td>I IN</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINE</td>
<td>I IT</td>
<td>50</td>
<td>φ</td>
<td>Line count</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
### VARIABLES

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMC</td>
<td>I</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
</tr>
<tr>
<td>LOCAL</td>
<td>I</td>
<td>O</td>
<td>XXXX</td>
<td>0</td>
<td>Local tax</td>
<td></td>
</tr>
<tr>
<td>LYNHR</td>
<td>I</td>
<td>O</td>
<td>0</td>
<td>0</td>
<td>This year's accumulation of hours worked for vacation pay</td>
<td></td>
</tr>
<tr>
<td>MAR</td>
<td>I</td>
<td>I, O</td>
<td>2</td>
<td>1</td>
<td>Marital status—(1-single), (2-married)</td>
<td></td>
</tr>
<tr>
<td>MUNC</td>
<td>I</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
</tr>
<tr>
<td>NADWH</td>
<td>I</td>
<td>O</td>
<td>XXXX</td>
<td>0</td>
<td>Additional withholding amount</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>A</td>
<td>9, I, O</td>
<td>-</td>
<td>-</td>
<td>Employee name</td>
<td></td>
</tr>
<tr>
<td>NCH:K</td>
<td>I</td>
<td>O</td>
<td>XXXX</td>
<td>0</td>
<td>Check number used for this employee</td>
<td></td>
</tr>
<tr>
<td>NC:U</td>
<td>I</td>
<td>I, O</td>
<td>XXXX</td>
<td>0</td>
<td>Credit union deduction</td>
<td></td>
</tr>
<tr>
<td>NC:U:O</td>
<td>I</td>
<td>I, O</td>
<td>XXXX</td>
<td>0</td>
<td>Monthly credit union deductions (in dimes)</td>
<td></td>
</tr>
<tr>
<td>NSD:ES</td>
<td>I</td>
<td>I, O</td>
<td>XXXX</td>
<td>0</td>
<td>Union dues deduction</td>
<td></td>
</tr>
<tr>
<td>NSD:K</td>
<td>A</td>
<td>2, I, O</td>
<td>-</td>
<td>-</td>
<td>Pay period date</td>
<td></td>
</tr>
<tr>
<td>NS:K:R</td>
<td>I</td>
<td>I, O</td>
<td>XXXX</td>
<td>0</td>
<td>Insurance deduction</td>
<td></td>
</tr>
<tr>
<td>NMS:CC</td>
<td>I</td>
<td>I, O</td>
<td>XXXX</td>
<td>0</td>
<td>Miscellaneous deductions</td>
<td></td>
</tr>
<tr>
<td>NOPLT</td>
<td>I</td>
<td>T</td>
<td>6</td>
<td>1</td>
<td>Plant number</td>
<td></td>
</tr>
<tr>
<td>NR:AT</td>
<td>I</td>
<td>I, O</td>
<td>3, 0.0</td>
<td>1.25</td>
<td>Employee pay rate</td>
<td></td>
</tr>
<tr>
<td>NSEX</td>
<td>I</td>
<td>I, O</td>
<td>3</td>
<td>1</td>
<td>Sex (2-female), (2-male), (3-trucker)</td>
<td></td>
</tr>
<tr>
<td>NSS:AN</td>
<td>I</td>
<td>3, I, O</td>
<td>Always</td>
<td>9</td>
<td>Social Security number</td>
<td></td>
</tr>
<tr>
<td>NST:AS</td>
<td>I</td>
<td>I, O</td>
<td>0</td>
<td>5</td>
<td>Employee status—(1-union), (2-trucker), (3-non-union, full-time), (4-non-union, part-time), (5-terminated)</td>
<td></td>
</tr>
<tr>
<td>NST:CK</td>
<td>I</td>
<td>I, O</td>
<td>XXXX</td>
<td>0</td>
<td>Stock deduction</td>
<td></td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSTKD</td>
<td>I</td>
<td>1</td>
<td>O</td>
<td>xx.xx</td>
<td>0</td>
</tr>
<tr>
<td>NUA</td>
<td>I</td>
<td>1</td>
<td>I,0</td>
<td>xx.xx</td>
<td>0</td>
</tr>
<tr>
<td>NUM</td>
<td>I</td>
<td>1</td>
<td>I,0</td>
<td>xxx</td>
<td>1000</td>
</tr>
<tr>
<td>NWKMP</td>
<td>I</td>
<td>1</td>
<td>I,0</td>
<td>xx</td>
<td>0</td>
</tr>
<tr>
<td>NWKPD</td>
<td>I</td>
<td>1</td>
<td>I,0</td>
<td>xx</td>
<td>0</td>
</tr>
<tr>
<td>NXMFP</td>
<td>I</td>
<td>1</td>
<td>I,0</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>NMXPS</td>
<td>I</td>
<td>1</td>
<td>I,0</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>OTERN</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
<tr>
<td>OTHER</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
<tr>
<td>OTHRS</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
<tr>
<td>QRTD</td>
<td>R</td>
<td>6/1</td>
<td>0</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
<tr>
<td>RSI</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
<tr>
<td>RGHR</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
<tr>
<td>SICK</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
<tr>
<td>SPA</td>
<td>R</td>
<td>3</td>
<td>T</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
<tr>
<td>SPB</td>
<td>R</td>
<td>3</td>
<td>T</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
<tr>
<td>SPECL</td>
<td>R</td>
<td>3</td>
<td>1</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
<tr>
<td>T</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
<tr>
<td>TAX</td>
<td>I</td>
<td>1</td>
<td>0</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
<tr>
<td>TAXBL</td>
<td>R</td>
<td>3</td>
<td>T</td>
<td>xxx.xxx</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**FUNCTION OF VARIABLES**

- **NSTKD**: Monthly stock deductions
- **NUA**: United Appeal deduction
- **NUM**: Clock number
- **NWKMP**: Number of weeks employed
- **NWKPD**: Number of weeks paid
- **NXMFP**: Federal exemptions
- **NMXPS**: State exemptions
- **OTERN**: Overtime earnings
- **OTHER**: Special earnings
- **OTHRS**: Overtime hours
- **QRTD**: Quarter-to-date information, (1) gross, (2) FII, (3) FICA, (4) loc. tax, (5) FICA wages, (6) sick pay
- **RSI**: Regular earnings
- **RGHR**: Regular hours
- **SICK**: Sick pay
- **SPA**: Special earnings accum. per ind.
- **SPB**: Special earnings accum. per ind.
- **SPECL**: Special earnings
- **T**: Used to total special earnings
- **TAX**: Federal withholding tax
- **TAXBL**: Taxable earnings

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
<table>
<thead>
<tr>
<th>VARIABLES</th>
<th>IBM 1130 COMPUTING SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VARIABLE SUMMARY SHEET</strong></td>
<td><strong>APPLICATION</strong> PAYROLL SYSTEM <strong>DATE</strong> 6/23/67</td>
</tr>
<tr>
<td><strong>PROGRAM NAME</strong> Calculations &amp; PR <strong>NO.</strong> PAY94 <strong>RICK</strong></td>
<td></td>
</tr>
</tbody>
</table>

**FUNCTION OF VARIABLES**

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGRS</td>
<td>R3T</td>
<td>4 &amp; 8xx</td>
<td>T</td>
<td>0.00</td>
<td>0.00</td>
<td>Total gross</td>
</tr>
<tr>
<td>TNET</td>
<td>R3T</td>
<td>4 &amp; 8xx</td>
<td>T</td>
<td>0.00</td>
<td>0.00</td>
<td>Total net</td>
</tr>
<tr>
<td>TOT</td>
<td>R2I</td>
<td>0 &amp; 8xx</td>
<td>O</td>
<td>0.00</td>
<td>0.00</td>
<td>Total array</td>
</tr>
<tr>
<td>TOTBN</td>
<td>R3I</td>
<td>4 &amp; 8xx</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>Bonus hour total from source doc.</td>
</tr>
<tr>
<td>TOTOT</td>
<td>R3I</td>
<td>4 &amp; 8xx</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>OT hour total from source doc.</td>
</tr>
<tr>
<td>TOTRG</td>
<td>R3I</td>
<td>4 &amp; 8xx</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>Reg. hour total from source doc.</td>
</tr>
<tr>
<td>TOTSP</td>
<td>R3I</td>
<td>4 &amp; 8xx</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>Special earnings total from source doc.</td>
</tr>
<tr>
<td>VACA</td>
<td>R3O</td>
<td>0 &amp; 8xx</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>Vacation pay</td>
</tr>
<tr>
<td>XBN</td>
<td>R3O</td>
<td>0 &amp; 8xx</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>Bonus hours error total</td>
</tr>
<tr>
<td>XOT</td>
<td>R3O</td>
<td>0 &amp; 8xx</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>Overtime hours error total</td>
</tr>
<tr>
<td>XREG</td>
<td>R3O</td>
<td>0 &amp; 8xx</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>Reg. hours error total</td>
</tr>
<tr>
<td>XSP</td>
<td>R3O</td>
<td>0 &amp; 8xx</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>Special earnings</td>
</tr>
<tr>
<td>YTD</td>
<td>R10</td>
<td>10 &amp; 8xx</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>Year-to-date information, (1) gross, (2) FICA, (3) FICA wages, (4) sick pay, (5) spec. A, (6) spec. B, (7) loc. tax, (8) reg. hrs, (9) OT hrs, (10) bonus hrs, (11) reg. erns, (12) OT erns, (13) bonus erns.</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
// FOR PAYO4
* I0CS(CARD+TYPEWRITER+KEYBOARD+1132 PRINTER+DISK)
* LIST ALL
* ** PAYO4 PROGRAM
* ** NAME PAYO4
* ** ONE WORD INTEGERS
* ** EXTENDED PRECISION
C----- JOB NAME -- PAYROLL SYSTEM - CALCULATIONS + PAYROLL REGISTER
C----- JOB NUMBER -- PAYO4
C----- PROGRAMMER -- C.R.KLICK
C----- DATE CODED -- 01/13/68
C----- DATE UPDATED --

<table>
<thead>
<tr>
<th>FILE</th>
<th>NAME</th>
<th>NUMBER LENGTH RECORD NO. OF RECORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT FILES</td>
<td>1. COLFP</td>
<td>1 160 250 2</td>
</tr>
<tr>
<td></td>
<td>2. WVAFP</td>
<td>2 160 200 2</td>
</tr>
<tr>
<td></td>
<td>3. MNCFP</td>
<td>3 160 200 2</td>
</tr>
<tr>
<td></td>
<td>4. LBOPP</td>
<td>4 160 50 2</td>
</tr>
<tr>
<td></td>
<td>5. LBTFP</td>
<td>5 160 150 2</td>
</tr>
<tr>
<td></td>
<td>6. LMCMP</td>
<td>6 160 30 2</td>
</tr>
<tr>
<td></td>
<td>7. PINFO</td>
<td>25 106 6 3</td>
</tr>
<tr>
<td></td>
<td>8. INDX1</td>
<td>102 1 250 320</td>
</tr>
<tr>
<td></td>
<td>9. INDX2</td>
<td>103 1 90 320</td>
</tr>
<tr>
<td></td>
<td>10. INDX5</td>
<td>104 1 200 320</td>
</tr>
<tr>
<td></td>
<td>11. INDX6</td>
<td>105 1 150 320</td>
</tr>
<tr>
<td></td>
<td>12. INDX8</td>
<td>106 1 30 320</td>
</tr>
</tbody>
</table>

| OUTPUT FILES | 1. COLFP | 1 160 250 2 |
| | 2. WVAFP | 2 160 90 2 |
| | 3. MNCFP | 3 160 200 2 |
| | 4. LBOPP | 4 160 50 2 |
| | 5. LBTFP | 5 160 150 2 |
| | 6. LMCMP | 6 160 30 2 |
| | 7. PINFO | 25 106 6 3 |

allocate array storage.
INTEGER COMP(16), TAX
DIMENSION FIBRE(8), IDATE(3), INDEX(250), ISUPP(13), ITOT(11),
1 KODE(3), NAME(9), NDWK(3), NSSAN(3), QRTD(6), SPECL(3),
2 TOT(21), YTD(14)

allocate array storage.

define the files for this program as described above, and
allocate the variables for next record number.

define file 1250*160(u;icol), 2190*160(u;wva),
1 3(220*160(u;munp), 4(150*160(u;ldp),
```plaintext
PAY04 PROGRAM

2 5(150+160+6+LBT)+6(30+150+106+L+1C)+25(6+106+U+IC): PAY04
• 3 101(250+U+IN1)+102(90+U+IN2)+103(200+U+IN3): PAY04
• 4 104(50+U+IN4)+105(150+U+IN5)+106(30+U+IN6): PAY04
• EQUIVALENCE (ICOL IWAV MUNC LBO LBT LMC): PAY04
• 1 (IN1 IN2 IN3 IN4 IN5 IN6): PAY04
• DEFINE AN ARITHMETIC STATEMENT FOR HALF ADJUSTING.
• PHIL(BET)=WHOLE((BET+5)/100): PAY04
• INITIALIZE VARIABLES
• ICOL=1
  IN1=1
• T=0
  XTOT=0
  XBN=0
• XREG=0
  XSP=0
• DO 50 I=1,21
  50 TOT(I)=0
• IPAGE=0
  L/NE=50
• READ PLANT NUMBER, DATE, AND CONTROL TOTALS
  99999 READ(2,1) NOPLT, IDATE, NOWK, TOTRG, TOTOT, TOTBN, TOTSP, KARD
  1 FORMAT(I11,6A2,7X,4F7.0,13X,I11)
• VALIDATE KARD AND NOPLT.
  IF VALID = 50
• IF INVALID = 55
• IF(KARD) 55,51,55
  51 IF(NOPLT) 55,55,52
  52 IF(NOPLT=6) 60,60,55
• FIRST CARD IS INVALID.
  WRITE(1,2)
  2 FORMAT('CHECK CC 1 AND 80 ON FIRST CARD')
  PAUSE 1
  GO TO 99999
• READ THE PLANT INFORMATION RECORD FROM DISK.
```
PAY04 PROGRAM

C-----
60 READ(25'NOPLT) COMP*, ICHCK*, IWEK*, FIBRE*, ITOT*, CKMAX
C-----
C----- WRITE THE PLANT INFORMATION FOR CONTROL PURPOSES AND ACCEPT ANY
C----- CHANGES TO IT THRU DATA SWITCH SETTINGS.
C-----
62 WRITE(I+31) COMP*, IDATE*, ICHCK*, IWEK*, NDWK*, CKMAX
3 FORMAT('16A2/3A2/"CHECK NO. "11/"WEEK NO. "11/"W/E "3A2/"NET MAX"
1 F6.0/"MAXIMUM CHECK AMOUNT MAY BE CHANGED BY SWITCH 14."
2 / "SWITCH 15 WILL CHANGE THE CHECK NO AND THE WEEK NO. SET"
3 "SWITCHES'/REQUESTED AND PRESS START'"
PAUSE 1111
CALL DATSW(15*1)
GO TO (70,71,1)
70 WRITE(1*4)
4 FORMAT('ENTER CHECK NO., FIVE DIGITS')
READ(I+22) ICHCK
WRITE(I+23)
23 FORMAT('ENTER WEEK NO., ONE DIGIT')
READ(I+24) IWEK
24 FORMAT(I1)
GO TO 62
71 CALL DATSW(14*1)
72 WRITE(I+25)
25 FORMAT('ENTER MAXIMUM CHECK AMOUNT, FIVE DIGITS')
READ(I+21) CKMAX
21 FORMAT(F5.0)
GO TO 62
C-----
C----- INITIALIZE PLANT VARIABLES
C-----
75 INDX=NOPLT + 100
GO TO (76*77*78*80+81)*NOPLT
76 ILST=250
GO TO 83
77 ILST=90
GO TO 83
78 ILST=200
GO TO 83
79 ILST=50
GO TO 83
80 ILST=150
GO TO 83
81 ILST=30
C-----
PAY04 PROGRAM

C----- READ THE EMPLOYEE INDEX FOR THIS PLANT.

83 READ(INDEX'ILST) LAST

C----- READ A WEEKLY EMPLOYEE RECORD.

90 READ(2,5) KPLNT, ICLK, RGRS, OTHRS, BNHRS, (KODE(I),SPECL(I))

C----- INITIALIZE INDIVIDUAL VARIABLES

ADREG=0.
AD=0.
HOLDY=0.
IFILL=0.
KO=16448
OTHER=0.
SICK=0.
SPA=0.
SPB=0.
TAX=0.
VACA=0.

C----- LAST CARD CHECK AND VALIDATE PLANT NUMBER

IF(KARD=6) 100,110,103
100 IF(KARD=9) 100,500,100

C----- PLANT NUMBER

110 IF(KPLNT = NOPLT) 100,105,100
100 WRITE(4,6) KPLNT, ICLK
6 FORMAT('CHECK CARD WITH CLOCK NUMBER '11.'13)
PAUSE 100
GO TO 90

C----- LOCATE EMPLOYEE IN INDEX

105 ICLK=ICLK + KPLNT * 1000
PAY04 PROGRAM

DO 115 IND=1, LAST
  IF(IINDEX(IND) .eq. ICLCK) 115, 125, 115
115 CONTINUE
C------
  PROGRAM COMES THRU HERE ONLY WHEN NO MATCH FOUND.
C------
WRITE(1, 17) ICLCK
7 FORMAT('CLOCK NO 'I4' IS NOT IN THE FILE')
C------
C------ UPDATE ERROR TOTALS
C------
120 XREG=XREG + RGHRS
  XTOT=XTOT + OTHRS
  XBNXBN + BNRS
  XSP=XSP + SPECL(1) + SPECL(2) + SPECL(3)
  CALL STACK
  GO TO 90
C------
C------ READ THE EMPLOYEE RECORD FROM DISK AND VALIDATE CLOCK NUMBER.
125 READ(NOPLT'IND) NUM, NAME, NSSAN, NSTAS, NDUES, NWKMP, NWKPD, MAR
2      NXMPF, NXMPS, NSEX, NRATE, YID, ORTD, LYRHR, NCU
3      NCUDD, NCHCK, NSTK, NINS, NMISC, NUA
C------
C------ VALIDATE CLOCK NUMBER
C------
135 IF(NUM .eq. ICLCK) 135, 136, 135
C------
136 WRITE(1, 8) NUM, ICLCK
8 FORMAT('FILE NO 'I4' AND INDEX NO 'I4' DO NOT AGREE')
GO TO 120
C------
C------ CALCULATE REGULAR EARNINGS AND HALF ADJUST
C------
136 RGERN=PHILIRGHRS * NRATE)
C------
C------ CALCULATE BONUS EARNINGS AND HALF ADJUST
C------
136 BNERN=PHIL1BNHRS * NRATE)
C------
C------ CALCULATE ANY SPECIAL EARNINGS; USE CODE TO DETERMINE TYPE OF
C------
17 EARNINGS.
    KODE  TYPE  KODE  TYPE
     1  SPA      5  SPB*NRATE
PAY04 PROGRAM

C=====

2 SPB 6 VACA

C=====

3 SPB*NRATE 7 SICK

C=====

4 SPB*NRATE 8 HOLDY

C=====

5 HOLDY * 2

DO 139 I=1,3

K=KODE(I)

IF(K) 100,139,600

600 GO TO (601,602,603,604,605,606,607,608,609),K

601 AD=SPECI(I)

OTHER=OTHER + AD

SPA=SPA + AD

K0=-3776

GO TO 139

602 OTHER=OTHER + SPECI(I)

SPB=SPA + SPECI(I)

K0=-3520

GO TO 139

603 K0=-3264

610 OTHER=PHI(SPECI(I) * NRATE)

SPB=SPB + SPECI(I)

GO TO 139

604 K0=-3008

GO TO 610

605 K0=-2752

GO TO 610

606 VACA=SPECI(I)

SPB=SPB + VACA

GO TO 139

607 SICK=SPECI(I)

GO TO 139

608 HOLDY=8. * NRATE

AD=AD + HOLDY

611 SPB=SPB + HOLDY

ADREG=800.

GO TO 139

609 HOLDY=16. * NRATE

AD=AD + HOLDY / 2.

GO TO 611

139 CONTINUE

C=====

CALCULATE OVERTIME EARNINGS IF APPLICABLE. USE STATUS AND HOURS

C=====

TO DETERMINE APPLICABILITY.

C=====

IF(NSTAS=2) 141,137,141

C=====

NOT APPLICABLE. USE STANDARD RATE.

C=====

88
PAY04 PROGRAM

137 IOTRT=NRATE
• GO TO 150 PAY04
• C======
141 IF(RGHR5) 137=137+142 PAY04
• C======
141 OVERTIME APPLIES. CALCULATE OVERTIME RATE.
• C======
142 IOTRT=(RGERN + BNERN + AD) * 100. / (RGHRS + ADREG) + 0.5 PAY04
• C======
143 CALCULATE OVERTIME PAY
• C======
150 OTERN=PHIL(OTHRS * IOTRT)
• C======
• C======
C====== SUM REGULAR, O.T., AND BONUS EARNINGS
• C======
ERNGS=RGERN + BNERN + OTERN
• C======
• C======
C====== CALCULATE AVERAGE RATE AND UPDATE LAST QUARTER AVERAGES.
• C======
IF(RGHR5) 143=143+144 PAY04
143 IVRAT=NRATE
GO TO 160 PAY04
144 IVRAT=ERNGS * 100. / RGHR5
160 DO 165 I=1,12 PAY04
165 ISUPP(I)=ISUPP(I+1)
ISUPP(13)=IOTRT
• C======
C======
C====== CALCULATE FICA TAXABLE EARNINGS
• C======
ERNGS=ERNGS + VACA + HOLDY + OTHER
• C======
C======
C====== CALCULATE FICA AND GROSS PAY AND TAXABLE PAY
• C======
IFICA=0.044 * ERNGS + 0.5 PAY04
IF(IFICA + YTD(2) = 29040., 185+180)+180
180 IFICA=29040. = YTD(2)
185 GROSS=ERNGS + SICK
• C======
C======
C======
C====== TAXBL=GROSS - NMHP * 1350.
• C======
C======
C======
C====== CALCULATE FEDERAL INCOME TAX
• C======
CALL IT(TAXBL, MAR, TAX)
• C======
CALL IT(TAXBL, MAR, TAX)
PAY04 PROGRAM

1. TAX = TAX + NADWH
2. COMPUTE LOCAL TAX BY PLANT LOCATION
3. GO TO (230+235+240+230+246+230) + NOPLT
4. LOCAL = PHIL(GROSS)
5. GO TO 250
6. 230 I + 0.2 = NXMPS + 0.5
7. LOCAL = 0.0108 * (GROSS - I)
8. GO TO 250
9. 240 IF(NXMPS) 241, 241, 242
10. LOCAL = 0.02 * GROSS
11. GO TO 250
12. 241 I = NXMPS + 1538.5 + 961.5
13. LOCAL = (GROSS - I) * 0.02
14. GO TO 250
15. 242 I = NXMPS + 1538.5 + 961.5
16. LOCAL = (GROSS - I) * 0.02
17. GO TO 250
18. 243 IF(I = LOCAL) 244, 245
19. LOCAL = 0
20. GO TO 250
21. 244 ATAX = GROSS - TAX - LOCAL
22. IFICA
23. CALCULATE VOLUNTARY DEDUCTIONS
24. INITIALIZE
25. IUD = 0
26. IUU = 0
27. ISTCK = 0
28. IN5 = 0
29. ICIU = 0
30. IMISC = 0
31. IF THE EMPLOYEE RECEIVES SICK PAY, VOLUNTARY DEDUCTIONS ARE NOT TAKEN
32. IF(SICK) 252, 252
33. CNET = ATAX
34. GO TO 35
35. OTHERWISE, DEDUCTIONS NECESSARY ARE TAKEN
36. TAKE UNION DUES ACCORDING TO PLANT
37. IF(IWEEK = 3) 255, 255
38. 252 I = IWEEK - 3, 255
39. GO TO 282
40. I = I + 1
41. 255 IF(INP = 2) 260, 260
42. 255 IF(NSTAS = 2) 260, 260
43. 260 IF(GROSS - VACA) 261, 261
PAY04 PROGRAM

261 GO TO (265+265+275+265+280),NOPLT
265 IF(NDUES - 10000) 270,270 NOPLT
270 IUD=NDUES + INIT
NDUES=NDUES + INIT + 10000
INIT=0
GO TO 290
275 IUD=PHIL(GROSS - VACA) + INIT
NDUES=NDUES + IUD
INIT=0
GO TO 282
280 IUD=0

282 IF(.WEEK - 2) 290,285 NOPLT
285 IF(NUA - 10000) 286,290 NOPLT
286 IUA=NUA
NUA=NUA + 10000
GO TO 295
290 IUA=0

295 ISTCK=NSTCK
IINS=NINS
ICU=NCU
IMISC=NMISC

300 IF(NOPLT - 3) 305,301
301 NDUES=NDUES - 10000
GO TO 309
305 NDUES=NDUES - 10000
309 IUD=0

310 CNET=ATAX - IUD
IF(CNET = IINS) 320,325,325
320 IINS=0
IFILL=1
325 CNET=CNET - IINS
IF(CNET = ISTCK) 330,335,335
330 ISTCK=0
IFILL=3
335 CNET=CNET - ISTCK
NSTKD=NSTKD + ISTCK
IF(CNET = ICU) 340,345,345

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>340</td>
<td>ICU=0</td>
</tr>
<tr>
<td>345</td>
<td>IFILL=4</td>
</tr>
<tr>
<td>346</td>
<td>ICU=CNET - ICU</td>
</tr>
<tr>
<td>347</td>
<td>NCUD=NCUD + (ICU / 10)</td>
</tr>
<tr>
<td>350</td>
<td>IUAA=0</td>
</tr>
<tr>
<td>355</td>
<td>IFillin=9</td>
</tr>
<tr>
<td>356</td>
<td>NUA=NUA - 10000</td>
</tr>
<tr>
<td>360</td>
<td>IIMISCI=IMISC</td>
</tr>
<tr>
<td>365</td>
<td>CNET=CNET - IMISC</td>
</tr>
<tr>
<td>366</td>
<td>IF(CNET - IMISC)</td>
</tr>
<tr>
<td>370</td>
<td>TAX=TAX + CNET</td>
</tr>
<tr>
<td>375</td>
<td>YTD(1)=YTD(1) + GROSS</td>
</tr>
<tr>
<td>376</td>
<td>YTD(2)=YTD(2) + IFICA</td>
</tr>
<tr>
<td>377</td>
<td>YTD(3)=YTD(3) + TAX</td>
</tr>
<tr>
<td>378</td>
<td>YTD(4)=YTD(4) + ERNGS</td>
</tr>
<tr>
<td>379</td>
<td>YTD(5)=YTD(5) + SICK</td>
</tr>
<tr>
<td>380</td>
<td>YTD(6)=YTD(6) + SPA</td>
</tr>
<tr>
<td>381</td>
<td>YTD(7)=YTD(7) + SPB</td>
</tr>
<tr>
<td>382</td>
<td>YTD(8)=YTD(8) + LOCAL</td>
</tr>
<tr>
<td>383</td>
<td>YTD(9)=YTD(9) + RGRHS</td>
</tr>
<tr>
<td>384</td>
<td>YTD(10)=YTD(10) + DTHRS</td>
</tr>
<tr>
<td>385</td>
<td>YTD(11)=YTD(11) + RHRRE</td>
</tr>
<tr>
<td>386</td>
<td>YTD(12)=YTD(12) + RGERN</td>
</tr>
<tr>
<td>387</td>
<td>YTD(13)=YTD(13) + DTERN</td>
</tr>
<tr>
<td>388</td>
<td>YTD(14)=YTD(14) + BNERN</td>
</tr>
<tr>
<td>389</td>
<td>YTD(15)=YTD(15) + OTHRS</td>
</tr>
<tr>
<td>390</td>
<td>YTD(16)=YTD(16) + BNHRS</td>
</tr>
<tr>
<td>391</td>
<td>YTD(17)=YTD(17) + RGERN</td>
</tr>
<tr>
<td>392</td>
<td>YTD(18)=YTD(18) + OTERN</td>
</tr>
<tr>
<td>393</td>
<td>YTD(19)=YTD(19) + GROSS</td>
</tr>
<tr>
<td>394</td>
<td>YTD(20)=YTD(20) + TAX</td>
</tr>
<tr>
<td>395</td>
<td>YTD(21)=YTD(21) + IFICA</td>
</tr>
<tr>
<td>396</td>
<td>YTD(22)=YTD(22) + TAX</td>
</tr>
<tr>
<td>397</td>
<td>YTD(23)=YTD(23) + ERNGS</td>
</tr>
<tr>
<td>398</td>
<td>YTD(24)=YTD(24) + SICK</td>
</tr>
<tr>
<td>399</td>
<td>YTD(25)=YTD(25) + SPA</td>
</tr>
<tr>
<td>400</td>
<td>YTD(26)=YTD(26) + SPB</td>
</tr>
<tr>
<td>401</td>
<td>YTD(27)=YTD(27) + LOCAL</td>
</tr>
<tr>
<td>402</td>
<td>YTD(28)=YTD(28) + RGRHS</td>
</tr>
<tr>
<td>403</td>
<td>YTD(29)=YTD(29) + DTHRS</td>
</tr>
<tr>
<td>404</td>
<td>YTD(30)=YTD(30) + RHRRE</td>
</tr>
<tr>
<td>405</td>
<td>YTD(31)=YTD(31) + RGERN</td>
</tr>
<tr>
<td>406</td>
<td>YTD(32)=YTD(32) + DTERN</td>
</tr>
<tr>
<td>407</td>
<td>YTD(33)=YTD(33) + BNERN</td>
</tr>
<tr>
<td>408</td>
<td>YTD(34)=YTD(34) + OTHRS</td>
</tr>
<tr>
<td>409</td>
<td>YTD(35)=YTD(35) + BNHRS</td>
</tr>
<tr>
<td>410</td>
<td>YTD(36)=YTD(36) + RGERN</td>
</tr>
<tr>
<td>411</td>
<td>YTD(37)=YTD(37) + OTERN</td>
</tr>
<tr>
<td>412</td>
<td>YTD(38)=YTD(38) + GROSS</td>
</tr>
<tr>
<td>413</td>
<td>YTD(39)=YTD(39) + TAX</td>
</tr>
<tr>
<td>414</td>
<td>YTD(40)=YTD(40) + IFICA</td>
</tr>
<tr>
<td>415</td>
<td>YTD(41)=YTD(41) + TAX</td>
</tr>
<tr>
<td>416</td>
<td>YTD(42)=YTD(42) + ERNGS</td>
</tr>
<tr>
<td>417</td>
<td>YTD(43)=YTD(43) + SICK</td>
</tr>
<tr>
<td>418</td>
<td>YTD(44)=YTD(44) + SPA</td>
</tr>
<tr>
<td>419</td>
<td>YTD(45)=YTD(45) + SPB</td>
</tr>
<tr>
<td>420</td>
<td>YTD(46)=YTD(46) + LOCAL</td>
</tr>
<tr>
<td>421</td>
<td>YTD(47)=YTD(47) + RGRHS</td>
</tr>
<tr>
<td>422</td>
<td>YTD(48)=YTD(48) + DTHRS</td>
</tr>
<tr>
<td>423</td>
<td>YTD(49)=YTD(49) + RHRRE</td>
</tr>
<tr>
<td>424</td>
<td>YTD(50)=YTD(50) + RGERN</td>
</tr>
<tr>
<td>425</td>
<td>YTD(51)=YTD(51) + DTERN</td>
</tr>
<tr>
<td>426</td>
<td>YTD(52)=YTD(52) + BNERN</td>
</tr>
<tr>
<td>427</td>
<td>YTD(53)=YTD(53) + OTHRS</td>
</tr>
<tr>
<td>428</td>
<td>YTD(54)=YTD(54) + BNHRS</td>
</tr>
<tr>
<td>429</td>
<td>YTD(55)=YTD(55) + RGERN</td>
</tr>
<tr>
<td>430</td>
<td>YTD(56)=YTD(56) + OTERN</td>
</tr>
<tr>
<td>431</td>
<td>YTD(57)=YTD(57) + GROSS</td>
</tr>
<tr>
<td>432</td>
<td>YTD(58)=YTD(58) + TAX</td>
</tr>
<tr>
<td>433</td>
<td>YTD(59)=YTD(59) + IFICA</td>
</tr>
<tr>
<td>434</td>
<td>YTD(60)=YTD(60) + TAX</td>
</tr>
<tr>
<td>435</td>
<td>YTD(61)=YTD(61) + ERNGS</td>
</tr>
<tr>
<td>436</td>
<td>YTD(62)=YTD(62) + SICK</td>
</tr>
<tr>
<td>437</td>
<td>YTD(63)=YTD(63) + SPA</td>
</tr>
<tr>
<td>438</td>
<td>YTD(64)=YTD(64) + SPB</td>
</tr>
<tr>
<td>439</td>
<td>YTD(65)=YTD(65) + LOCAL</td>
</tr>
<tr>
<td>440</td>
<td>YTD(66)=YTD(66) + RGRHS</td>
</tr>
<tr>
<td>441</td>
<td>YTD(67)=YTD(67) + DTHRS</td>
</tr>
<tr>
<td>442</td>
<td>YTD(68)=YTD(68) + RHRRE</td>
</tr>
<tr>
<td>443</td>
<td>YTD(69)=YTD(69) + RGERN</td>
</tr>
<tr>
<td>444</td>
<td>YTD(70)=YTD(70) + DTERN</td>
</tr>
<tr>
<td>445</td>
<td>YTD(71)=YTD(71) + BNERN</td>
</tr>
<tr>
<td>446</td>
<td>YTD(72)=YTD(72) + OTHRS</td>
</tr>
<tr>
<td>447</td>
<td>YTD(73)=YTD(73) + BNHRS</td>
</tr>
<tr>
<td>448</td>
<td>YTD(74)=YTD(74) + RGERN</td>
</tr>
<tr>
<td>449</td>
<td>YTD(75)=YTD(75) + OTERN</td>
</tr>
<tr>
<td>450</td>
<td>YTD(76)=YTD(76) + GROSS</td>
</tr>
<tr>
<td>451</td>
<td>YTD(77)=YTD(77) + TAX</td>
</tr>
<tr>
<td>452</td>
<td>YTD(78)=YTD(78) + IFICA</td>
</tr>
<tr>
<td>453</td>
<td>YTD(79)=YTD(79) + TAX</td>
</tr>
<tr>
<td>454</td>
<td>YTD(80)=YTD(80) + ERNGS</td>
</tr>
<tr>
<td>455</td>
<td>YTD(81)=YTD(81) + SICK</td>
</tr>
<tr>
<td>456</td>
<td>YTD(82)=YTD(82) + SPA</td>
</tr>
<tr>
<td>457</td>
<td>YTD(83)=YTD(83) + SPB</td>
</tr>
<tr>
<td>458</td>
<td>YTD(84)=YTD(84) + LOCAL</td>
</tr>
<tr>
<td>459</td>
<td>YTD(85)=YTD(85) + RGRHS</td>
</tr>
<tr>
<td>460</td>
<td>YTD(86)=YTD(86) + DTHRS</td>
</tr>
<tr>
<td>461</td>
<td>YTD(87)=YTD(87) + RHRRE</td>
</tr>
<tr>
<td>462</td>
<td>YTD(88)=YTD(88) + RGERN</td>
</tr>
<tr>
<td>463</td>
<td>YTD(89)=YTD(89) + DTERN</td>
</tr>
<tr>
<td>464</td>
<td>YTD(90)=YTD(90) + BNERN</td>
</tr>
<tr>
<td>465</td>
<td>YTD(91)=YTD(91) + OTHRS</td>
</tr>
<tr>
<td>466</td>
<td>YTD(92)=YTD(92) + BNHRS</td>
</tr>
<tr>
<td>467</td>
<td>YTD(93)=YTD(93) + RGERN</td>
</tr>
<tr>
<td>468</td>
<td>YTD(94)=YTD(94) + OTERN</td>
</tr>
<tr>
<td>469</td>
<td>YTD(95)=YTD(95) + GROSS</td>
</tr>
<tr>
<td>470</td>
<td>YTD(96)=YTD(96) + TAX</td>
</tr>
</tbody>
</table>
PAY04 PROGRAM

QRTD(3) = QRTD(3) + IFICA
QRTD(4) = QRTD(4) + LOCAL
QRTD(5) = QRTD(5) + ERNGS
QRTD(6) = QRTD(6) + SICK

C------ UPDATE PLANT TOTALS
C------
TOT(1) = TOT(1) + RGHRS
TOT(2) = TOT(2) + RGERN
TOT(3) = TOT(3) + OTHRHS
TOT(4) = TOT(4) + OTERN
TOT(5) = TOT(5) + BNHRS
TOT(6) = TOT(6) + BNERN
TOT(7) = TOT(7) + OTHER
TOT(8) = TOT(8) + HOLDY
TOT(9) = TOT(9) + VACA
TOT(10) = TOT(10) + SICK
TOT(11) = TOT(11) + CNET
TOT(12) = TOT(12) + TAX
TOT(13) = TOT(13) + IFICA
TOT(14) = TOT(14) + LOCAL
TOT(15) = TOT(15) + ICU
TOT(16) = TOT(16) + IUD
TOT(17) = TOT(17) + IUA
TOT(18) = TOT(18) + ISTCK
TOT(19) = TOT(19) + IMISC
TOT(20) = TOT(20) + IINS
TOT(21) = TOT(21) + GROSS

C------ SUM SPECIAL EARNINGS, SUM DEDUCTIONS, AND EXTEND THE EMPLOYEE WEEKLY CARD
C------
T = T + SPECL(1) + SPECL(2) + SPECL(3)
IDED + IINS + ISTCK + IUA + IMISC
WRITE(2+9), NRATE, GROSS, CNET, TAX, IFICA, LOCAL, ICU, IUD, IDED
9 FORMAT(17X, I3, 2F6.0, 15, 414, I5)

C------ SETUP CONTROL INFORMATION, AND WRITE UPDATED EMPLOYEE RECORD BACK TO THE DISK.
C------
LYHR = LYHR + RGHRS
NWKP = NWKP + 1
IPD = 1

C------ WRITE(NOPT+1ND), NUM, NAME, SSN, NSTAS, NONES, NWKMP, NWKP, I
1 MAR, NWKP, NKMPS, NSEX, NRATE, YTD, QRTD, LLYHR, NCU, NCUD, PAY04
PAY04 PROGRAM

PAY04

2 NCHCK, NADWH, NSTCK, NINS, NMISC, NUA, NSTKD, ISupp, INIT.

3 IPD, IFILL, GROSS, IVRAT, IOTRT, RGHRs, OTHERs, BNHRs, RGERN.

4 OTHERs, BNERN, OTHER+ KO, HOLDY, VACA, SICK, CNET, IFICA.

5 LOCAL, ICU, IUA, IUD, IINS, ISTCK, IMISC.

C——— GO BACK FOR ANOTHER WEEKLY EMPLOYEE CHECK.

C——— GO TO 90

C——— WRITE 1ST PAYROLL REGISTER.

500 ICNT=1 CHCK

DO 510 I=1, LAST

READ(NOPLT'I) NUM, NAME, SSN, STAS, DUES, NWMP, NWKD, MAR.

1 NXMFP, NXMPS, NSEX, NRAT, YTD, QT, LYHR, NCU, NCUD.

C——— CHECK PAID INDICATOR TO SEE IF COMPUTATIONS WERE PERFORMED.

IF(IPD - 1) 510, 515, 510

515 RGHRs=WHOLE(RGHRs + (RGHRs / ABS(RGHRs)) * 5/100).

OTHERs=WHOLE(OTHERs + (-OTHERs / ABS(-OTHERs)) * 5/100).

BNHRs=WHOLE(BNHRs + (BNHRs / ABS(BNHRs)) * 5/100).

RGERN=WHOLE(RGERN + (RGERN / ABS(RGERN)) * 5/100).

OTHER+=WHOLE(OTHER+ + (-OTHER+ / ABS(-OTHER+)) * 5/100).

HOLDY=WHOLE(HOLDY + (HOLDY / ABS(HOLDY)) * 5/100).

VACA=WHOLE(VACA + (VACA / ABS(VACA)) * 5/100).

SICK=WHOLE(SICK + (SICK / ABS(SICK)) * 5/100).

GROSS=WHOLE(GROSS + (GROSS / ABS(GROSS)) * 5/100).

NET=WHOLE(CNET + (CNET / ABS(CNET)) * 5/100).

IF(LINE - 50) 385, 380, 380

380 IPAGE=IPAGE + 1

WRITE(3, 19) FACTORY PAYROLL, PAGE NO 012/12.

19 FORMAT('LINE','11X0', 'NAME', 1I6, 'REG HRS OT HRS BNS HRS REG ERN OT BNS ERN SPECIAL HOLDAY VACA SICK GROSS')

WRITE(3, 20)

20 FORMAT('FICA FW T LOCAL C+U, U/D U/A INS STCK MISC NET')

WRITE(3, 11) NUM, NAME, ICNT, RGHRs, OTHERs, BNHRs, RGERN, OTHER+

1 BNERN, KO, OTHER+ HOLDY, VACA, SICK, GROSS, IFICA.
PAY04 PROGRAM

2       TAX, LOCAL, ICU, IUD, IUAA, IINS, ISTCK, IMISC, CNET PAY04
11      FORMAT(/01X,I4,14X,92A15/6(2X,F6.2)/1X,A15(12X,F6.2)/1X,A15,3X,F8.2)/PAY04
1       FORMAT(/01X,I4,14X,92A15/6(2X,F6.2)/PAY04
FIBRE(NSEX)=FIBRE(NSEX) + 1 PAY04
LINE=LINE + 3 PAY04
ICNT=ICNT + 1 PAY04
510 CONTINUE PAY04
C----- WRITE CONTROL TOTALS PAY04
C----- WRITE THE PLANT GENERAL LEDGER INFORMATION AFTER THE TOTAL LINE PAY04
C----- WRITE THE PLANT INFORMATION BACK TO DISK PAY04

C----- WRITE CONTROL TOTALS PAY04
TGRS=TOT(21) PAY04
TNET=TOT(21) PAY04
WRITE(1,15) TGRS, TOT(1), TOT(3), TOT(5), TOT(8) PAY04
15 FORMAT('INPUT TOTALS ',4(F8.0,3X)) PAY04
WRITE(1,16) TOT(1), TOT(3), TOT(5), TOT(8) PAY04
16 FORMAT('PROCESSED TOTALS ',4(F8.0,3X)) PAY04
WRITE(1,17) TOT(1), TOT(3), TOT(5), TOT(8) PAY04
17 FORMAT('ERROR TOTALS ',4(F8.0,3X)) PAY04
A=TGRS - TOT(1) - XREG PAY04
B=TOTOT - TOT(3) - XTOT PAY04
C=TOTBN - TOT(5) - XBN PAY04
D=TOTSP - TOT(8) - XSP PAY04
WRITE(1,18) A, B, C, D PAY04
18 FORMAT('THE DIFFERENCES',4(F8.0,3X)) PAY04
C----- WRITE THE PLANT GENERAL LEDGER INFORMATION AFTER THE TOTAL LINE PAY04
C----- WRITE THE PLANT INFORMATION BACK TO DISK PAY04

C----- WRITE THE PLANT GENERAL LEDGER INFORMATION AFTER THE TOTAL LINE PAY04
FIBRE(3)=FIBRE(3) + TOT(1) PAY04
FIBRE(4)=FIBRE(4) + TOT(2) PAY04
FIBRE(5)=FIBRE(5) + TOT(3) PAY04
FIBRE(6)=FIBRE(6) + TOT(4) PAY04
FIBRE(7)=FIBRE(7) + TOT(5) PAY04
FIBRE(8)=FIBRE(8) + TOT(6) PAY04
520 TOT(1)=WHOLE(TOT(1)) - (TOT(1) / ABS(TOT(1))) * 0.5 / 100. PAY04
WRITE(3,19) TOT(1), TOT(1), TOT(1) PAY04
19 FORMAT(/020X,I4,5X,F9.2) PAY04
DO 550 I=1,10 PAY04
550 WRITE(3,14) TOT(I), TOT(I+10) PAY04
14 FORMAT(/020X,I4,5X,F9.2) PAY04
C----- WRITE THE PLANT INFORMATION BACK TO DISK PAY04
C----- WRITE THE PLANT INFORMATION BACK TO DISK PAY04

CALLED SUBPROGRAMS

INTEGER CONSTANTS

REAL CONSTANTS

STATEMENT ALLOCATIONS

VARIABLE ALLOCATIONS

CORE REQUIREMENTS FOR PAY04

CORE WORD CONSTANTS

CORE WORD INTEGERS

CORE REQUIREMENTS FOR Pay04

COMMON D VARIABLES

END OF COMPIlATION

PAY04 PROGRAM

C--

CALL EXIT

STOP

PAY04

PAGE 14

PAY04
THE CONTAINER CORP. 022168
CHECK NO  1
WEEK NO  1
W/E 021568
NET MAX 25000.

MAXIMUM CHECK AMOUNT MAY BE CHANGED BY SWITCH 14.
SWITCH 15 WILL CHANGE THE CHECK NO AND THE WEEK NO. SET SWITCHES
REQUESTED AND PRESS START

CHECK CARD WITH CLOCK NUMBER 1  6
INPUT TOTALS  44000.  1650.  1050.  13900.
PROCESSED TOTALS  40000.  1650.  1050.  12700.
ERROR TOTALS  0.  0.  0.  0.
THE DIFFERENCES  4000.  0.  0.  1200.
Test output

| NUMBR | NAME          | REG HRS | OT HRS | BNS  | HRS | REG ENR | OT ENR | BNS ENR | SPECIAL HOLIDAY | VACATION | SICK | FICA | FWT | LOCAL | C.U. | U/D  | U/A | INS | STCK | MISC | NET |     |
|-------|---------------|---------|--------|------|-----|---------|--------|--------|----------------|-----------|------|------|-----|-------|-----|------|-----|-----|------|------|-----|-----|     |
| 1001  | ROBERT BADEN  | 40.00   | 0.00   | 1.00 | 104.40 | 0.00   | 2.61 | 12.00 | 0.00         | 0.00       | 0.00 | 0.00 | 119.01 |
| 1002  | JOHN A MORN   | 40.00   | 0.00   | 1.00 | 104.40 | 0.00   | 0.00 | 5.00  | 10.44         | 0.00       | 0.00 | 0.00 | 114.84 |
| 1003  | ROBERT SNORES | 40.00   | 2.50   | 0.00 | 85.60  | 5.35   | 0.00 | 4.00  | 8.56          | 0.00       | 0.00 | 0.00 | 99.51 |
| 1004  | JOHN W CUSSEN | 40.00   | 0.00   | 0.00 | 104.40 | 0.00   | 0.00 | 5.00  | 10.44         | 0.00       | 0.00 | 0.00 | 114.84 |
| 1005  | JOSEPH MONTANO| 40.00   | 3.00   | 2.00 | 146.80 | 11.73  | 7.44 | 29.76 | 0.00          | 3.00       | 0.00 | 0.00 | 200.73 |
| 1016  | DONALD MILLER | 40.00   | 5.00   | 0.00 | 112.00 | 14.00  | 0.00 | 0.00  | 14.00         | 0.00       | 0.00 | 0.00 | 132.00 |
| 1107  | ANDREW S TAYLOR| 40.00   | 0.00   | 2.50 | 104.40 | 0.00   | 6.52 | 20.88 | 0.00          | 8.00       | 0.00 | 0.00 | 139.80 |
| 1218  | DAVID A HUBBARD| 40.00   | 5.00   | 0.00 | 85.60  | 12.50  | 0.00 | 0.00  | 24.24         | 0.00       | 0.00 | 0.00 | 132.24 |
| 1347  | FRANK T OLEEN | 40.00   | 5.00   | 0.00 | 68.40  | 0.00   | 5.13 | 7.00  | 27.36         | 0.00       | 0.00 | 0.00 | 107.89 |
| 1603  | AL REYNOLDS   | 40.00   | 1.00   | 2.00 | 148.80 | 4.01   | 7.44 | 2.00  | 8.56          | 0.00       | 0.00 | 0.00 | 164.25 |

FST LINE TOTAL 400.00 1066.80 18.50 47.59 10.50 29.14 84.20 82.48 19.00 12.00
<table>
<thead>
<tr>
<th>Section</th>
<th>Subsections</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

**FACTORY PAYROLL**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>-996.85</td>
<td></td>
</tr>
<tr>
<td>620</td>
<td>-159.84</td>
<td></td>
</tr>
<tr>
<td>620</td>
<td>-58.49</td>
<td></td>
</tr>
<tr>
<td>422</td>
<td>-19.36</td>
<td></td>
</tr>
<tr>
<td>625</td>
<td>-22.50</td>
<td></td>
</tr>
<tr>
<td>626</td>
<td>-34.50</td>
<td></td>
</tr>
<tr>
<td>627</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>628</td>
<td>-5.00</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-50.67</td>
<td></td>
</tr>
<tr>
<td>635</td>
<td>1341.21</td>
<td></td>
</tr>
</tbody>
</table>

Printer output, part 2
IBM 1130 MACHINE SETUP SHEET

PROGRAM NAME: Calculations & Payroll Register
PROGRAM NUMBER: PAY04
PROGRAM DESCRIPTION: APPROXIMATE RUNNING TIME:

<table>
<thead>
<tr>
<th>PRINTER</th>
<th>TYPE OF PAPER</th>
<th>NO. OF COPIES</th>
<th>CARRIAGE TAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>1</td>
<td>Standard</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISKS</th>
<th>DRIVE NUMBER:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CARTRIDGE ID:</td>
<td>Payroll</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWITCH SETTINGS</th>
<th>SWITCH 14</th>
<th>SWITCH 15</th>
<th>SWITCH None</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOWN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

INPUT CARDS: Switch 14 to change maximum check amount (and turn off) Switch 15 to change check number and week number (and turn off)

SOURCE OF INPUT: 1. Card input from a successful PAY16 edit run. 2. Disk must be payroll disk from files.


FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS
### VARIABLES

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>Min.</th>
<th>Max.</th>
<th>Function of Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>R3</td>
<td>0</td>
<td>0.00</td>
<td>1.00</td>
<td>used for zero balance check</td>
</tr>
<tr>
<td>B</td>
<td>R3</td>
<td>0</td>
<td>0.00</td>
<td>1.00</td>
<td>used for zero balance check</td>
</tr>
<tr>
<td>ENEIY</td>
<td>R3</td>
<td>0</td>
<td>0.00</td>
<td>1.00</td>
<td>Bonus earnings</td>
</tr>
<tr>
<td>ENEIY</td>
<td>R3</td>
<td>0</td>
<td>0.00</td>
<td>1.00</td>
<td>Bonus hours</td>
</tr>
<tr>
<td>BR</td>
<td>R3</td>
<td>0</td>
<td>0.00</td>
<td>1.00</td>
<td>Print out of maximum check amount</td>
</tr>
<tr>
<td>CKMAX</td>
<td>R1</td>
<td>0</td>
<td>0.00</td>
<td>1.00</td>
<td>Maximum check amount for a file</td>
</tr>
<tr>
<td>CNET</td>
<td>R3</td>
<td>0</td>
<td>0.00</td>
<td>1.00</td>
<td>Net amount of individual check</td>
</tr>
<tr>
<td>COMP</td>
<td>A2</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>Company name</td>
</tr>
<tr>
<td>FIBRE</td>
<td>R3</td>
<td>0</td>
<td>0.00</td>
<td>1.00</td>
<td>Trade assoc. reports</td>
</tr>
<tr>
<td>GROSS</td>
<td>R3</td>
<td>0</td>
<td>0.00</td>
<td>1.00</td>
<td>Gross amount of individual check</td>
</tr>
<tr>
<td>HOLDS</td>
<td>R3</td>
<td>0</td>
<td>0.00</td>
<td>1.00</td>
<td>Individual's holiday pay</td>
</tr>
<tr>
<td>I</td>
<td>I/I</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>Used in DO loop</td>
</tr>
<tr>
<td>IC</td>
<td>I/N</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>JCHK</td>
<td>I/I</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>Beginning check number when writing check</td>
</tr>
<tr>
<td>ICNT</td>
<td>I/I</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>Sequence number for journal (should correspond to check #)</td>
</tr>
<tr>
<td>ICOL</td>
<td>I/T</td>
<td>250</td>
<td>1</td>
<td></td>
<td>Record number by employee file, set up by plant</td>
</tr>
<tr>
<td>ICU</td>
<td>I/I</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>Individual's credit union deduction</td>
</tr>
<tr>
<td>IDATE</td>
<td>A2</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>Pay date</td>
</tr>
<tr>
<td>ID1</td>
<td>I/I</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>1st check number</td>
</tr>
<tr>
<td>ID2</td>
<td>I/I</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>1st check number</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT</th>
<th>OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFICA</td>
<td>I/O</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Individual's FICA tax</td>
</tr>
<tr>
<td>IFILL</td>
<td>I/T</td>
<td>7</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Indicates deduction not made</td>
</tr>
<tr>
<td>IINS</td>
<td>I/O</td>
<td>XX</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Individual's insurance deduction</td>
</tr>
<tr>
<td>ILST</td>
<td>I/T</td>
<td>250</td>
<td></td>
<td></td>
<td>50</td>
<td></td>
<td>Last record number in a file</td>
</tr>
<tr>
<td>IMISC</td>
<td>I/O</td>
<td>XXXX</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Individual's misc. deductions</td>
</tr>
<tr>
<td>INDX</td>
<td>I/T</td>
<td>106</td>
<td></td>
<td></td>
<td>101</td>
<td></td>
<td>Index file number (plant no + 100)</td>
</tr>
<tr>
<td>INIT</td>
<td>I/O</td>
<td>XXXX</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Union initiation fee</td>
</tr>
<tr>
<td>IN1</td>
<td>I/T</td>
<td>250</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>Record number in indexes to employees file</td>
</tr>
<tr>
<td>IN2</td>
<td>I/N</td>
<td>–</td>
<td></td>
<td></td>
<td>–</td>
<td>–</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN3</td>
<td>I/N</td>
<td>–</td>
<td></td>
<td></td>
<td>–</td>
<td>–</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN4</td>
<td>I/N</td>
<td>–</td>
<td></td>
<td></td>
<td>–</td>
<td>–</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN5</td>
<td>I/N</td>
<td>–</td>
<td></td>
<td></td>
<td>–</td>
<td>–</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN6</td>
<td>I/N</td>
<td>–</td>
<td></td>
<td></td>
<td>–</td>
<td>–</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>I01RT</td>
<td>I/T</td>
<td>500</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Overtime pay rate</td>
</tr>
<tr>
<td>IPD</td>
<td>I/O</td>
<td>2</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Indicates status of record in processing cycle</td>
</tr>
<tr>
<td>IPNT</td>
<td>I/T</td>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>Set by data switch, controls pauses between prints</td>
</tr>
<tr>
<td>ISTCK</td>
<td>I/O</td>
<td>2000</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Individual's stock deduction</td>
</tr>
<tr>
<td>ISPUP</td>
<td>I/O</td>
<td>XXXX</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Supplemental sick pay</td>
</tr>
<tr>
<td>ITOT</td>
<td>I/T</td>
<td>1723</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td>Account number for posting to general ledger</td>
</tr>
<tr>
<td>IUA</td>
<td>I/O</td>
<td>300</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Individual's charity deduction</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
## VARIABLES

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP</th>
<th>OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IUD</td>
<td>I</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Individual's union dues deduction</td>
</tr>
<tr>
<td>IVRT</td>
<td>I</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Average pay rate</td>
</tr>
<tr>
<td>IWK</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>5</td>
<td>1</td>
<td></td>
<td>Week of the month</td>
</tr>
<tr>
<td>IWA</td>
<td>I</td>
<td>N</td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
<td>Equivalent to DCOL</td>
</tr>
<tr>
<td>II</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>Xxxxx</td>
<td></td>
<td></td>
<td>Convert regular hours to printable form</td>
</tr>
<tr>
<td>I2</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>Xxxxx</td>
<td></td>
<td></td>
<td>Convert overtime hours to printable form</td>
</tr>
<tr>
<td>I3</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>Xxxxx</td>
<td></td>
<td></td>
<td>Convert bonus hours to printable form</td>
</tr>
<tr>
<td>I4</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>Xxxxx</td>
<td></td>
<td></td>
<td>Convert regular earnings to printable form</td>
</tr>
<tr>
<td>I5</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>Xxxxx</td>
<td></td>
<td></td>
<td>Convert overtime earnings to printable form</td>
</tr>
<tr>
<td>I6</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>Xxxxx</td>
<td></td>
<td></td>
<td>Convert bonus earnings to printable form</td>
</tr>
<tr>
<td>I7</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>Xxxxx</td>
<td></td>
<td></td>
<td>Convert other earnings to printable form</td>
</tr>
<tr>
<td>I8</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>Xxxxx</td>
<td></td>
<td></td>
<td>Convert holiday pay to printable form</td>
</tr>
<tr>
<td>I9</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>Xxxxx</td>
<td></td>
<td></td>
<td>Convert sick pay to printable form</td>
</tr>
<tr>
<td>JGRS</td>
<td>A</td>
<td>1</td>
<td>T</td>
<td>00000</td>
<td></td>
<td></td>
<td>For moved gross</td>
</tr>
<tr>
<td>JOUT1</td>
<td>A</td>
<td>5</td>
<td>0</td>
<td>XXXXX</td>
<td></td>
<td></td>
<td>For edited vacation pay</td>
</tr>
<tr>
<td>JOUT2</td>
<td>A</td>
<td>1</td>
<td>T</td>
<td>00000</td>
<td></td>
<td></td>
<td>For edited gross pay</td>
</tr>
<tr>
<td>JVAC</td>
<td>A</td>
<td>5</td>
<td>T</td>
<td>XXXXX</td>
<td></td>
<td></td>
<td>For moved vacation</td>
</tr>
<tr>
<td>KARO</td>
<td>I</td>
<td>1</td>
<td>I</td>
<td>9</td>
<td></td>
<td></td>
<td>c.c. 80 for last card test</td>
</tr>
<tr>
<td>KO</td>
<td>A</td>
<td>1</td>
<td>O</td>
<td>5</td>
<td></td>
<td></td>
<td>Special earnings</td>
</tr>
<tr>
<td>LAST</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>XXXXX</td>
<td></td>
<td></td>
<td>Last record number in file</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
### VARIABLES

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE*</th>
<th>No. of Words</th>
<th>INPUT/TEMP</th>
<th>OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBO</td>
<td>I  I</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LBT</td>
<td>I  I</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMC</td>
<td>I  I</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCAL</td>
<td>I  I</td>
<td>O</td>
<td>XXXX</td>
<td>ø</td>
<td>local tax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LVRHR</td>
<td>I  I</td>
<td>O</td>
<td>XXXX</td>
<td>ø</td>
<td>This year's accumulation of hours worked for welfare pay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAR</td>
<td>I  I,</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>Marital status - (1 = single), (2 = married)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MASK</td>
<td>A  T</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>Edit mask (X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MASK2</td>
<td>A  T</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>Edit mask (zero suppress)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUNC</td>
<td>I  I</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NADWH</td>
<td>I  I</td>
<td>O</td>
<td>XXXX</td>
<td>ø</td>
<td>Additional withholding amount</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>A  9</td>
<td>I, I</td>
<td>-</td>
<td>-</td>
<td>Employee name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCCHK</td>
<td>I  I</td>
<td>O</td>
<td>XXXX</td>
<td>ø</td>
<td>Check number used for this employee</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCU</td>
<td>I  I</td>
<td>O</td>
<td>XXXX</td>
<td>ø</td>
<td>Credit union deduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCUDO</td>
<td>I  I</td>
<td>O</td>
<td>XXXX</td>
<td>ø</td>
<td>Monthly credit union deductions (in dimes)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDUES</td>
<td>I  I,</td>
<td>0</td>
<td>XX, XX</td>
<td>ø</td>
<td>China dues deduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDWK</td>
<td>A  3</td>
<td>I, I</td>
<td>-</td>
<td>-</td>
<td>Pay period date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NET</td>
<td>A  T</td>
<td>70</td>
<td>$XXXX</td>
<td>$.00</td>
<td>Edited net</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NET1</td>
<td>A  T</td>
<td>70</td>
<td>XXXX</td>
<td>-</td>
<td>Edited net</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NET2</td>
<td>A  T</td>
<td>70</td>
<td>XXXX</td>
<td>-</td>
<td>Edited net</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NET4</td>
<td>A  T</td>
<td>70</td>
<td>XXXX</td>
<td>ø</td>
<td>Moved net</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic

---

### IBM 1130 COMPUTING SYSTEM

**VARIABLE SUMMARY SHEET**

- **Application:** PAYROLL SYSTEM
- **Date:** 9/6/67
- **Program Name:** Check Writing
- **No.:** PAY05
- **Programmer:** KIT"
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP</th>
<th>OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NM5</td>
<td>I</td>
<td>10</td>
<td>XX,XX</td>
<td></td>
<td>0</td>
<td></td>
<td>Insurance deduction</td>
</tr>
<tr>
<td>NMISC</td>
<td>I</td>
<td>10</td>
<td>XXXX</td>
<td></td>
<td>0</td>
<td></td>
<td>Miscellaneous deductions</td>
</tr>
<tr>
<td>NUPIT</td>
<td>I I T</td>
<td>6</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>Plant number</td>
</tr>
<tr>
<td>NRATE</td>
<td>I I D</td>
<td>3,00</td>
<td>1.25</td>
<td></td>
<td></td>
<td></td>
<td>Employee pay rate</td>
</tr>
<tr>
<td>NRITE</td>
<td>I I T</td>
<td>XXXX</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Clock number for repeat printing</td>
</tr>
<tr>
<td>NSEX</td>
<td>I I O</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>Sex-(1-female), (2-male), (3-trucker)</td>
</tr>
<tr>
<td>NSSAN</td>
<td>I 3</td>
<td>I O</td>
<td>Always 9</td>
<td>digits</td>
<td></td>
<td></td>
<td>Social Security number</td>
</tr>
<tr>
<td>NSTAS</td>
<td>I O</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>Employee status-(1-union), (2-trucker), (3-nonunion fulltime), (4-non-union part-time), (5-terminated)</td>
</tr>
<tr>
<td>NSTCK</td>
<td>I I O</td>
<td>XXXX</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Stock deduction</td>
</tr>
<tr>
<td>NSTKD</td>
<td>I O</td>
<td>XXXX</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Monthly stock deductions</td>
</tr>
<tr>
<td>NUA</td>
<td>I I O</td>
<td>XXXX</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>United Appeal deduction</td>
</tr>
<tr>
<td>NUM</td>
<td>I I O</td>
<td>XXXX</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Clock number</td>
</tr>
<tr>
<td>NWKMP</td>
<td>I O</td>
<td>XX</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Number of weeks employed</td>
</tr>
<tr>
<td>NWKPD</td>
<td>I O</td>
<td>XX</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Number of weeks paid</td>
</tr>
<tr>
<td>NXMPE</td>
<td>I I O</td>
<td>17</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Federal exemptions</td>
</tr>
<tr>
<td>NXMPS</td>
<td>I I O</td>
<td>17</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>State exemptions</td>
</tr>
<tr>
<td>OTERN</td>
<td>R</td>
<td>30</td>
<td>00</td>
<td></td>
<td>00</td>
<td></td>
<td>Overtime earnings</td>
</tr>
<tr>
<td>OTHER</td>
<td>R</td>
<td>30</td>
<td>00</td>
<td></td>
<td>00</td>
<td></td>
<td>Special earnings</td>
</tr>
<tr>
<td>OTHRS</td>
<td>R</td>
<td>30</td>
<td>00</td>
<td></td>
<td>00</td>
<td></td>
<td>Overtime hours</td>
</tr>
<tr>
<td>QRND</td>
<td>R/8</td>
<td>0</td>
<td>00</td>
<td></td>
<td>00</td>
<td></td>
<td>Quarter-to-date information:(4)gross,(2)FIT (3)FICA,(4)act. tax,(5)FICA wages,(6) sick pay</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP</th>
<th>OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRN</td>
<td>R 3 0</td>
<td>XXXX</td>
<td>3</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Regular earnings</td>
</tr>
<tr>
<td>RGHR5</td>
<td>R 3 0</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Regular hours</td>
</tr>
<tr>
<td>SICK</td>
<td>R 3 0</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Sick pay</td>
</tr>
<tr>
<td>TA</td>
<td>R 3 0</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Total gross by company</td>
</tr>
<tr>
<td>TAX</td>
<td>I 1 0</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Federal Withholding Tax</td>
</tr>
<tr>
<td>TB</td>
<td>R 3 0</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Total net by company</td>
</tr>
<tr>
<td>TGRS</td>
<td>R 3 T</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Total gross</td>
</tr>
<tr>
<td>TNET</td>
<td>R 3 T</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Total net</td>
</tr>
<tr>
<td>TOTBN</td>
<td>R 3 I</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Bonus hours total from source doc.</td>
</tr>
<tr>
<td>TOTOT</td>
<td>R 3 I</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>OT hours total from source doc.</td>
</tr>
<tr>
<td>TOTRG</td>
<td>R 3 I</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Reg. hours total from source doc.</td>
</tr>
<tr>
<td>TOTSD</td>
<td>R 3 I</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Special earnings total from source doc.</td>
</tr>
<tr>
<td>VACA</td>
<td>R 3 0</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Vacation pay</td>
</tr>
<tr>
<td>YIN1</td>
<td>A 1 7</td>
<td>T</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Moved gross YTD</td>
</tr>
<tr>
<td>YIN2</td>
<td>A 1 6</td>
<td>T</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Moved Federal Tax YTD</td>
</tr>
<tr>
<td>YOUT1</td>
<td>A 1 7</td>
<td>O</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Edited gross YTD</td>
</tr>
<tr>
<td>YOUT2</td>
<td>A 1 6</td>
<td>O</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Edited Federal Tax YTD</td>
</tr>
<tr>
<td>YTD</td>
<td>R 4 0</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>Year-to-date information: (1) gross, (2) FICA, (3) FIT, (4) FICA wages, (5) sick pay, (6) spec A, (7) spec B, (8) loc tax, (9) Reg hours, (10) OT hours, (11) bonus hours, (12) Reg bonus, (13) OT bonus, (14) bonus bonus.</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
11
// FOR 10CSICARD, TYPEWRITER, KEYBOARD, 1132 PRINTER DISK1
** PAY05 PROGRAM
* NAME PAY05
* ONE WORD INTEGERS
* EXTENDED PRECISION
C----- JOB NAME = PAYROLL SYSTEM CHECK WRITING
C----- JOB NUMBER = PAY05
C----- PROGRAMMER = C.R. KLICK
C----- DATE CODED = 01/20/68
C----- DATE UPDATED =
C----- FILE NAME NUMBER LENGTH RECORDS PAY05
C----- INPUT FILES --
1. COLFP 1 160 250 2 PAY05
2. WVAFP 2 160 90 2 PAY05
3. MNCFP 3 160 200 2 PAY05
4. LBTFP 5 160 150 2 PAY05
5. LMCFP 6 160 30 2 PAY05
7. PINFO 25 106 6 3 PAY05
8. INDX1 101 1 250 320 PAY05
9. INDX2 102 1 90 320 PAY05
10. INDX3 103 1 200 320 PAY05
11. INDX4 104 1 90 320 PAY05
12. INDX5 105 1 150 320 PAY05
13. INDX6 106 1 30 320 PAY05
C----- OUTPUT FILES --
1. COLFP 1 160 250 2 PAY05
2. WVAFP 2 160 90 2 PAY05
3. MNCFP 3 160 200 2 PAY05
4. LBTFP 5 160 150 2 PAY05
5. LMCFP 6 160 30 2 PAY05
7. PINFO 25 106 6 3 PAY05
C----- ALLOCATE ARRAY STORAGE PAY05
C----- INTEGER COMP(16), TAX, YIN1(7), YIN2(6), YOUT1(7), YOUT2(6)
C----- DIMENSION FIBRE(8), IDATE(3), ISUPP(13), ITOT(11), JGROS(7)
1 JOUT1(5), JOUT2(7), JVACA(5), MASK(7), MASK2(7)
2 NAME(9), NDWK(3), NETO(7), NET1(7), NET2(7), NET4(7)
3 N55ANI3(7), ORTO(6), YTD(14)
C----- DEFINE FILES FOR THIS PROGRAM AS DESCRIBED ABOVE AND EQUIVALENCE PAY05
C----- THE VARIABLES FOR THE NEXT RECORD NUMBER
C----- DEFINE FILE 1(250,160,U+ICOL), 2(190,160,U+IWVA)
PAY05 PROGRAM

```plaintext
1 3(200,160,U,MUNC), 4(50,160,U,LBO)
2 101(250,1,U,IN1), 102(90,10,U,IN2), 103(2001,1,U,IN3), PAY05
3 104(50,1,U,IN4), 105(150,10,U,IN5), 106(30,1,U,IN6)
4 EQUIVALENCE (ICOL,INVA,MUNC,LBO,LBT,LMC),
5 (IN1,IN2,IN3,IN4,IN5,IN6)

C----- INITIALIZE VARIABLES
C----- DO 4 I=1,7
4 MASK(I)=16448
4 MASK(7)=4032
C----- ICOL=1
C----- IN1=1
C----- TA=0.
C----- TB=0.
C----- NRITE=0

C----- READ PLANT NO., DATE, AND CONTROL TOTALS, AND VALIDATE CC 80 AND PAY05
C----- THE PLANT NUMBER.
C----- 99999 READ(2,1) NOPLT, MATE, NDWK, TOTRG, TOTOT, TOTBN, TOTSP, KARD
1 FORMAT(I16A2,4F7.0,38X,I11)
C----- VALIDATE KARD AND NOPLT
C----- IF KARD 55,51.55
C----- IF NOPLT 55,51.
C----- IF INVALID - 55
C----- IF(KARD) 55,51,55
C----- IF(NOPLT) 55,51,55
C----- IF(NOPLT = 6) 60,60,55
C----- WRITE(I1,2)
2 FORMAT('CHECK CC 1 AND CC 80 ON FIRST CARD')
PAUSE 1
GO TO 99999
C----- READ THE PLANT INFORMATION RECORD FROM DISK.
C----- 60 READ(25'NOPLT) COMP, ICHECK, IWKW, FIGRE, ITOT, CKMAX, TGRS, TNET
C----- ICNT
C----- WRITE THE PLANT INFORMATION FOR CONTROL PURPOSES AND ACCEPT ANY
```

105
C---- CHANGES TO IT THRU DATA SWITCH SETTINGS.

C----
62 WRITE(1,13) COMP, IDATE, ICHCK, IWEK, NDWK, CKMAX
3 FORMAT (/16A2,'3A2','CHECK NO 'I5,'WEEK NO 'I1,'WE '3A2,' CHECK PAY05
1 MAX,'F8.0,'MAXIMUM CHECK AMOUNT MAY BE CHANGED BY SWITCH 14/.'PAY05
2SWITCH 15 WILL CHANGE THE CHECK NUMBER'/ 'SET SWITCHES REQUESTED ANPAY05
3P8.0/0MAXIMUM CHECK AMOUNT MAY BE CHANGED BY SWITCH 14/.'PAY05

30 PRESS START'

PAUSE 1111

CALL DSW(15,I)

GO TO (7001).1

71 CALL DSW(14,I)

GO TO (7205)11

72 WRITE(1,21)

READ(6,22) ICHCK

21 FORMAWENTER CHECK NO - FIVE DIGITS'

22 FORMAT (15)

GO TO 62

71 CALL DSW(14,I)

GO TO (7205)11

72 WRITE(1,23)

23 FORMAT ('ENTER MAXIMUM CHECK AMOUNT - FIVE DIGITS')

READ(6,24) CKMAX

24 FORMAT (F5.0)

GO TO 62

C---- COMPLETE VARIABLE INITIALIZATION

C----
75 INDX=NOPLT + 100

GO TO (76,77,78,79,80,81,NOPLT)

76 ILST=251

GO TO 83

77 ILST=90

GO TO 83

78 ILST=260

GO TO 83

79 ILST=30

GO TO 83

80 ILST=190

GO TO 83

81 ILST=30

C---- READ AN EMPLOYEE RECORD FROM DISK, AND USE THE PAID INDICATOR TO

C---- DECIDE IF A CHECK SHOULD BE WRITTEN.

C----
83 READ(INDX',ILST), LAST

ICHEC=ICHCK - 1

870 DO 700 I=1,LAST

READ(NOPLT), NUM, NAME, NSSAN, NSTAS, NDUES, NKMP, NWKPD, MAR,

1, NMPF, NXMP, NSEX, NRATE, YTD, ORTD, LENGTH, NCU, NCUD

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05

PAY05
PAY05 PROGRAM

90 PAUSE 2

C-----------
91 11=RGNRS / 10. + 0.05
12=OTHRS / 10. + 0.05
13=BNHRS / 10. + 0.05
14=RQNN
15=RQNN
16=BNERN
17=OTHER
18=HOLDY
19=SICK
CALL PUT(JVACA,+15<VACA * 10.+5.+1)
CALL PG(MASK2,17<JOUT1,11)
CALL MOVE(MASK2,17<JOUT2,11)
CALL EDIT(JVACA,15<JOUT1,11)
CALL EDIT(JVACA,15<JOUT2,11)
C-----------
92 PAUSE 3
C-----------
93 CALL DATSW(15,IPNT)
GO TO (92,93),IPNT
C-----------
94 PAUSE 4
PAY05 PROGRAM

C-----
95 CALL PUT(YIN1.1.7*YTD(1) = 10*5*1)
95 CALL PUT(YIN2.1.6*YTD(2) = 10*5*1)
95 CALL MOVE(MASK2.1.7*YOUT1.1)
95 CALL MOVE(MASK2.2.7*YOUT2.1)
95 CALL EDIT(YIN1.1.7*YOUT1.1)
95 CALL EDIT(YIN2.1.6*YOUT2.1)
95 ID1=YTD(1)
95 ID2=YTD(2)

C-----
WRITE FOURTH LINE OF CHECK AND GO BACK FOR ANOTHER EMPLOYEE.

C-----
WRITE(1,500) YOUT1, YOUT2, ID1, ID2
500 FORMAT(13A1,2I5)

C-----
CALL DATSW(15,IPNT)
GO TO (96,700).IPNT
96 PAUSE S

C-----
GO BACK
700 CONTINUE
C-----
WRITE 1V SPECIAL CHECKS. SIGNAL THIS CONDITION WITH DATA SWITCH 0.

C-----
CALL DATSW(0,1)
850 WRITE(1,25)
850 READ(6,26) WRITE
GO TO 870

C-----
WRITE CONTROL TOTALS

C-----
855 ICNT=ICNT - 1
800 IF(ICHCK = ICNT) 800,801,S800
800 WRITE(1,100) ICNT, ICHCK
100 FORMAT('REGISTER CHECK NO 'IS' DOES NOT AGREE WITH THIS RUN CHECK NO '15')
100 GO TO 802
801 WRITE(1,101)
101 FORMAT('CHECK NUMBERS AGREE')
802 A=GRS - TA
802 B=TNET - TB
802 WRITE(1,102) GRS, TNET, TA, TB
102 FORMAT(3X,F9.0)/ CHECK TOTALS '2(3X,F9.0)/ 109
PAY05 PROGRAM

1  'DIFFERENCES  '213X.F9.0, PAY05

• WRITE UPDATED PLANT RECORD TO DISK  PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05

• PAY05
// JOB
// XEQ PAY05 3
- *FILES(1*COLFP)*\{2*WVFAP\}*\{3*MNCFP\}*\{4*LBOFP\}*\{5*LBTFP\}*\{6*LMCFP\}*
- *FILES(25*PINFO)*
- *FILES(101*INDEX1)*\{102*INDEX2\}*\{103*INDEX3\}*\{104*INDEX4\}*\{105*INDEX5\}*\{106*INDEX6\}*
- 1022164021568 0040000000 16500000 105000012700

Input cards
### Payroll Statement

**The Container Company**

**To The National Bank & Trust Co. of Columbus, Wash.**

<table>
<thead>
<tr>
<th>Payroll Number</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1901</td>
<td>$86.08</td>
</tr>
<tr>
<td>1902</td>
<td>$83.55</td>
</tr>
</tbody>
</table>

**Check Details**

- **Check No. 35**
- **Check No. 45**
- **Check No. 55**

**Check Total:** $170.63

---

**Console Printer Output**

THE CONTAINER CORP. 022168

- Week No 1
- W/E 021568
- Check Max 25000.

MAXIMUM CHECK AMOUNT MAY BE CHANGED BY SWITCH 14.

SWITCH 15 WILL CHANGE THE CHECK NUMBER

SET SWITCHES REQUESTED AND PRESS START

CHECK NUMBERS AGREE

REGISTER TOTALS 134121. 99685.

CHECK TOTALS 134121. 99685.

DIFFERENCES 0. 0.
IBM 1130 MACHINE SETUP SHEET

PROGRAM NAME: Check Writing
PROGRAM NUMBER: PAY05

PROGRAM DESCRIPTION:

<table>
<thead>
<tr>
<th>TYPE OF PAPER</th>
<th>NO. OF COPIES</th>
<th>CARRIAGE TAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks</td>
<td></td>
<td>Checks</td>
</tr>
</tbody>
</table>

PRINTER

<table>
<thead>
<tr>
<th>DRIVE NUMBER:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARTRIDGE ID:</td>
<td>Payroll</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DISKS

SWITCH SETTINGS

<table>
<thead>
<tr>
<th>SWITCH</th>
<th>0</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOWN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

INPUT CARDS

Switch 0 is used to make checks reprint when they are not correct.
Switch 14 is used to set the maximum check amount.
Switch 15 is used to set the check number to start with, and to stop the system to align the printer.

SOURCE OF INPUT:
1. Control totals from file D
2. Disk must be payroll disk from file

DISPOSITION OF OUTPUT:
1. Paychecks to employees
2. Disk of control totals to be used with PAY06

FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS
| IBM 1403 Models | IBM 407, 408, 409, 1403, 1404, 1443, and 2203
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Printer Spacing Chart</td>
</tr>
<tr>
<td><strong>IBM 1403 Models</strong></td>
<td>8 Lines Per Inch</td>
</tr>
</tbody>
</table>

**Blocks**

**Factory Control**

Company Name is printed here

**WJ: 30-32-46**

**TOTAL:**

**CHECK REGISTER**
<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP</th>
<th>OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNEE</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>XXXX</td>
<td>.00</td>
<td>.00</td>
<td>Bonus earnings</td>
</tr>
<tr>
<td>BNHRS</td>
<td>R</td>
<td>3</td>
<td>I,0</td>
<td>XXXX</td>
<td>.00</td>
<td>.00</td>
<td>Bonus hours</td>
</tr>
<tr>
<td>CKNX</td>
<td>R</td>
<td>1/5</td>
<td>T</td>
<td>XXXX</td>
<td>.00</td>
<td>.00</td>
<td>Maximum check amount for a file</td>
</tr>
<tr>
<td>CNET</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>XXXX</td>
<td>.00</td>
<td>.00</td>
<td>Net amount of individual check</td>
</tr>
<tr>
<td>COMP</td>
<td>A</td>
<td>1/6</td>
<td>I,0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Company name</td>
</tr>
<tr>
<td>FIBRE</td>
<td>R</td>
<td>.2</td>
<td>0</td>
<td>XXXX</td>
<td>.00</td>
<td>.00</td>
<td>Trade association reports</td>
</tr>
<tr>
<td>GROSS</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>XXXX</td>
<td>.00</td>
<td>.00</td>
<td>Gross amount of individual check</td>
</tr>
<tr>
<td>HOLY</td>
<td>R</td>
<td>3</td>
<td>0</td>
<td>XX,XX</td>
<td>.00</td>
<td>.00</td>
<td>Individual's holiday pay</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>-</td>
<td>-</td>
<td>Used in DO loop</td>
</tr>
<tr>
<td>IC</td>
<td>I</td>
<td>I</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>ICHCK</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Beginning check number when writing checks</td>
</tr>
<tr>
<td>ICNT</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>Sequence number for journal (should correspond)</td>
</tr>
<tr>
<td>ICOL</td>
<td>I</td>
<td>I</td>
<td>T</td>
<td>250</td>
<td>1</td>
<td>1</td>
<td>Record number in employee files, set up by plant</td>
</tr>
<tr>
<td>ICU</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>Individual's credit union deduction</td>
</tr>
<tr>
<td>IDATE</td>
<td>I</td>
<td>I</td>
<td>3/4</td>
<td>0</td>
<td>XXXX</td>
<td>0</td>
<td>Total of individual's insurance, stock, charity, and misc. deductions per pay period</td>
</tr>
<tr>
<td>ID1</td>
<td>I</td>
<td>I</td>
<td>0</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>1st check number</td>
</tr>
<tr>
<td>ID2</td>
<td>I</td>
<td>I</td>
<td>0</td>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>1st clock number</td>
</tr>
<tr>
<td>ID3</td>
<td>I</td>
<td>3</td>
<td>A,0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>12th name</td>
</tr>
<tr>
<td>ID4</td>
<td>I</td>
<td>I</td>
<td>0</td>
<td>XXXX</td>
<td>0</td>
<td>2</td>
<td>2nd check number</td>
</tr>
<tr>
<td>ID5</td>
<td>I</td>
<td>I</td>
<td>0</td>
<td>XXXX</td>
<td>0</td>
<td>2</td>
<td>2nd clock number</td>
</tr>
</tbody>
</table>

*Modes: I = integer, R = real, D = decimal, A = alphabetic*
### VARIABLES

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP/OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDG</td>
<td>I/0</td>
<td>2</td>
<td></td>
<td>2nd name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID1</td>
<td>I/0</td>
<td>xxxxx</td>
<td></td>
<td>3rd check number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID8</td>
<td>I/0</td>
<td>xxxxx</td>
<td></td>
<td>3rd clock number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID9</td>
<td>I/0</td>
<td></td>
<td></td>
<td>3rd name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFICA</td>
<td>I/0</td>
<td>xxxxx</td>
<td></td>
<td>Individual's FICA tax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFILL</td>
<td>I/T</td>
<td>7</td>
<td></td>
<td>Indicated deduction not made</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN5</td>
<td>I/0</td>
<td>xx</td>
<td></td>
<td>Individual's insurance deduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILST</td>
<td>I/T</td>
<td>250</td>
<td></td>
<td>Last record number in a file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMISC</td>
<td>I/0</td>
<td>xxxxx</td>
<td></td>
<td>Individual's misc. deduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDX</td>
<td>I/T</td>
<td>106 191</td>
<td></td>
<td>Index file number (plant no. + 100)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INIT</td>
<td>I/0</td>
<td>xxxxx</td>
<td></td>
<td>Union initiation fee</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN1</td>
<td>I/T</td>
<td>250</td>
<td></td>
<td>Record number in indexes to employee files</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN2</td>
<td>I/N</td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN3</td>
<td>I/N</td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN4</td>
<td>I/N</td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN5</td>
<td>I/N</td>
<td></td>
<td></td>
<td>Equivalent to INZ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN6</td>
<td>I/N</td>
<td></td>
<td></td>
<td>Equivalent to IN1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOTRT</td>
<td>I/T</td>
<td>500</td>
<td></td>
<td>Overtime pay rate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPD</td>
<td>I/0</td>
<td>2</td>
<td></td>
<td>Indicates status of record in processing cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISTACK</td>
<td>I/0</td>
<td>2000</td>
<td></td>
<td>Individual's stock deduction</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>NO. OF WORDS</th>
<th>INPUT/TEMP OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISUPP</td>
<td>I/I</td>
<td>0</td>
<td></td>
<td>XXXX</td>
<td>0</td>
<td>Supplemental sick pay</td>
</tr>
<tr>
<td>ITOT</td>
<td>I/I</td>
<td>T</td>
<td></td>
<td>1723</td>
<td>0</td>
<td>Account number for posting to in general</td>
</tr>
<tr>
<td>IUH</td>
<td>I/I</td>
<td>0</td>
<td></td>
<td>300</td>
<td>0</td>
<td>Individual's charity deduction</td>
</tr>
<tr>
<td>IUO</td>
<td>I/I</td>
<td>0</td>
<td></td>
<td>1500</td>
<td>0</td>
<td>Individual's union dues deduction</td>
</tr>
<tr>
<td>IVRAT</td>
<td>I/I</td>
<td>0</td>
<td></td>
<td>500</td>
<td>0</td>
<td>Average pay rate</td>
</tr>
<tr>
<td>I WEEK</td>
<td>I/I</td>
<td>T</td>
<td></td>
<td>5</td>
<td>1</td>
<td>Week of the month</td>
</tr>
<tr>
<td>INVVA</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>J</td>
<td>I/I</td>
<td>T</td>
<td></td>
<td>9</td>
<td>1</td>
<td>Index for DO loop</td>
</tr>
<tr>
<td>KCARD</td>
<td>I/I</td>
<td>9</td>
<td></td>
<td>0</td>
<td>0</td>
<td>C.C. B for last Card test</td>
</tr>
<tr>
<td>KO</td>
<td>A/I</td>
<td>0</td>
<td></td>
<td>5</td>
<td>0</td>
<td>Special earnings code</td>
</tr>
<tr>
<td>L</td>
<td>I/I</td>
<td>T</td>
<td></td>
<td>250</td>
<td>0</td>
<td>Counter to access records</td>
</tr>
<tr>
<td>LAST</td>
<td>I/I</td>
<td>T</td>
<td></td>
<td>XXX</td>
<td>0</td>
<td>Last record number in file</td>
</tr>
<tr>
<td>LBO</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LBT</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LMC</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LOCAL</td>
<td>I/O</td>
<td>XXX</td>
<td></td>
<td>0</td>
<td>0</td>
<td>Local tax</td>
</tr>
<tr>
<td>Lyrhr</td>
<td>I/O</td>
<td>XXX</td>
<td></td>
<td>0</td>
<td>0</td>
<td>This year's accumulation of hours worked</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>for vacation pay</td>
</tr>
<tr>
<td>MAR</td>
<td>I/I</td>
<td>0</td>
<td></td>
<td>2</td>
<td>1</td>
<td>Marital status - (1. Single) (2. Married)</td>
</tr>
<tr>
<td>MUNC</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td>0</td>
<td>0</td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>NADWH</td>
<td>I/O</td>
<td>XXX</td>
<td></td>
<td>0</td>
<td>0</td>
<td>Additional withholding amount</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
# VARIABLES

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>I 2 9</td>
<td>I 0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>I</td>
<td>Employee name</td>
</tr>
<tr>
<td>NCHCK</td>
<td>I 1 0</td>
<td>I 0</td>
<td>xxxxx</td>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
<td>I</td>
<td>Check number used for this employee.</td>
</tr>
<tr>
<td>NCU</td>
<td>I 1 I 0</td>
<td>I 0</td>
<td>xx.xx</td>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
<td>I</td>
<td>Credit union deduction</td>
</tr>
<tr>
<td>NUDD</td>
<td>I 1 I 0</td>
<td>I 0</td>
<td>xxxxx</td>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
<td>I</td>
<td>Monthly credit union deductions</td>
</tr>
<tr>
<td>NDUES</td>
<td>I 1 I 0</td>
<td>I 0</td>
<td>xx.xx</td>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
<td>I</td>
<td>Union dues deduction</td>
</tr>
<tr>
<td>NOWK</td>
<td>I 2 3</td>
<td>I 0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>I</td>
<td>Pay period date</td>
</tr>
<tr>
<td>NINS</td>
<td>I 1 I 0</td>
<td>I 0</td>
<td>xx.xx</td>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
<td>I</td>
<td>Insurance deduction</td>
</tr>
<tr>
<td>NMISC</td>
<td>I 1 I 0</td>
<td>I 0</td>
<td>xxxxx</td>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
<td>I</td>
<td>Miscellaneous deductions</td>
</tr>
<tr>
<td>NOPLT</td>
<td>I 1 I 0</td>
<td>I 0</td>
<td>g 1</td>
<td>1</td>
<td>1</td>
<td>I</td>
<td>Plant number</td>
</tr>
<tr>
<td>NRATE</td>
<td>I 1 I 0</td>
<td>3.00</td>
<td>1.25</td>
<td>1.25</td>
<td>1.25</td>
<td>I</td>
<td>Employee pay rate</td>
</tr>
<tr>
<td>NSEX</td>
<td>I 1 I 0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>I</td>
<td>Sex (1-female), (2-male), (3-trucker)</td>
</tr>
<tr>
<td>NSSAN</td>
<td>I 3 I 0</td>
<td>Always</td>
<td>9 digits Social Security number</td>
<td>9 digits</td>
<td>9 digits</td>
<td>I</td>
<td>Social Security number</td>
</tr>
<tr>
<td>NSTAS</td>
<td>I 1 I 0</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>I</td>
<td>Employee status (1-union), (5-trucker) (6-non-union, fulltime), (7-non-union, parttime), (8-terminated)</td>
</tr>
<tr>
<td>NSTCK</td>
<td>I 1 I 0</td>
<td>I 0</td>
<td>xx.xx</td>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
<td>I</td>
<td>Stock deduction</td>
</tr>
<tr>
<td>NSTKD</td>
<td>I 1 I 0</td>
<td>I 0</td>
<td>xxxxx</td>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
<td>I</td>
<td>Monthly stock deductions</td>
</tr>
<tr>
<td>NOA</td>
<td>I 1 I 0</td>
<td>I 0</td>
<td>xx.xx</td>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
<td>I</td>
<td>United Appeal deductions</td>
</tr>
<tr>
<td>NUM</td>
<td>I 1 I 0</td>
<td>I 0</td>
<td>xxxx</td>
<td>1000</td>
<td>1000</td>
<td>I</td>
<td>Clock number</td>
</tr>
<tr>
<td>NWKMP</td>
<td>I 1 I 0</td>
<td>I 0</td>
<td>xx</td>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
<td>I</td>
<td>Number of weeks employed</td>
</tr>
<tr>
<td>NWKPD</td>
<td>I 1 I 0</td>
<td>I 0</td>
<td>xx</td>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
<td>I</td>
<td>Number of weeks paid</td>
</tr>
<tr>
<td>NXMPF</td>
<td>I 1 I 0</td>
<td>17</td>
<td></td>
<td>$\varnothing$</td>
<td>$\varnothing$</td>
<td>I</td>
<td>Federal exemptions</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INTERFACE</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NXMPS</td>
<td>I</td>
<td>10</td>
<td></td>
<td>0</td>
<td>0</td>
<td>State exemptions</td>
</tr>
<tr>
<td>OTERN</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Overtime Earnings</td>
</tr>
<tr>
<td>OTHER</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Special earnings</td>
</tr>
<tr>
<td>OTHERS</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Overtime hours</td>
</tr>
<tr>
<td>QRTD</td>
<td>R</td>
<td>10</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Quarter-to-date information (1) gross, (2) FIT, (3) FICA, (4) loc. tax, (5) FICA wages, (6) sick pay</td>
</tr>
<tr>
<td>RGERN</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Reg. earnings</td>
</tr>
<tr>
<td>RGQRS</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Reg. hours</td>
</tr>
<tr>
<td>RNET1</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>1st net</td>
</tr>
<tr>
<td>RNET2</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>2nd net</td>
</tr>
<tr>
<td>RNET3</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>3rd net</td>
</tr>
<tr>
<td>SICK</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Sick pay</td>
</tr>
<tr>
<td>T</td>
<td>I</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Used to total special earnings</td>
</tr>
<tr>
<td>TAX</td>
<td>I</td>
<td>10</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Federal Withholding Tax</td>
</tr>
<tr>
<td>TGRS</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Total gross</td>
</tr>
<tr>
<td>TNET</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Total net</td>
</tr>
<tr>
<td>TOTBN</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Bonus hours total from source doc.</td>
</tr>
<tr>
<td>TOTOT</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>OT hours total from source doc.</td>
</tr>
<tr>
<td>TOTRG</td>
<td>R</td>
<td>30</td>
<td></td>
<td>.00</td>
<td>.00</td>
<td>Reg. hours total from source doc.</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
## VARIABLE SUMMARY SHEET

**IBM 1130 COMPUTING SYSTEM**

### VARIABLES

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTSP</td>
<td>R I</td>
<td>8</td>
<td>XXX</td>
<td>XXXXX</td>
<td>0.00</td>
<td>Special earnings total from source doc.</td>
</tr>
<tr>
<td>VACA</td>
<td>R O</td>
<td>5</td>
<td>XXX</td>
<td>XXXXX</td>
<td>0.00</td>
<td>Vacation pay</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
Start

Initialize Variables

Read Plant No. Date and Control Totals

Are S0 and Plant No. Valid?

Yes

Read the Plant Inform. Record

Initialize Plant Variables

No

Put together Check Register Information

Write a Line of Check Register

Write Last Line of Check Register

Write the Plant Total

Last Employee?

Yes

Stop
// FOR
// IOCS(CARD,TYPEWRITER), 1132 PRINTER DISK

NAME PAY06
ONE WORD INTEGERS
EXTENDED PRECISION
LIST ALL

--- JOB NAME --- PAYROLL SYSTEM CHECK REGISTER
--- JOB NUMBER --- PAY06
--- PROGRAMER --- CR*KLICK
--- DATE CODED --- 01/27/68
--- DATE UPDATED ---

--- FILE FILE RECORD NO. OF RECORDS PER SECTOR

--- INPUT FILES ---
1. COLFP 1 160 250 2
2. WVAFP 2 160 90 2
3. MNCFP 3 160 200 2
4. LB0FP 4 160 50 2
5. LBTFP 5 160 150 2
6. LMCFP 6 160 30 2
7. PINFO 25 106 6 3
8. INDEX1 101 1 250 320
9. INDEX2 102 1 90 320
10. INDEX3 103 1 200 320
11. INDEX4 104 1 50 320
12. INDEX5 105 1 150 320
13. INDEX6 106 1 30 320

--- OUTPUT FILES ---
NONE

--- ALLOCATE ARRAY STORAGE
INTEGER COMP(161), TAX
DIMENSION FIBRE(8), IDATE(3), ID3(9), ID6(9), ISUPP(13)
ITOT(11), NAME(9), NOWK(3), NSSAN(3), ORTD(6), YTD(14)

--- DEFINE THE FILES FOR THIS PROGRAM AS DESCRIBED ABOVE, AND EQUIVALENCE THE VARIABLES FOR THE NEXT RECORD NUMBER.


--- INITIALIZE VARIABLES

122
C-----
• ICOL=1
• IN1=1
• TB=0.

C-----
• READ PLANT NO, DATE, AND CONTROL TOTALS, AND VALIDATE CC 80 AND

C-----
• 99999 READ(2+1) NOPLT, IDATE, NDWK, TOTRG, TOTOT, TOTBN, TOTSP, KARD

C-----
• IF (KARD) 55:51:55
• IF (NOPLT) 55:55:52
• IF (NOPLT = 6) 60:60:55

C-----
• WRITE(1,2)
• 2 FORMAT('CHECK CC 1 AND CC 80 ON FIRST CARD')

C-----
• IF NOPLT — 6) 60,60,155

C-----
• GO TO 99999

C-----
• READ PLANT INFORMATION RECORD FROM DISK, AND FINISH INITIALIZING.

C-----
• 60 READ(25'NOPLT) COMP, ICHCK, IWEEK, FIBRE,icot, CKMAX, TGRS, TNET, ICNT

C-----
• INDX=NOPLT + 100
• GO TO (76+77+78+79+80+81)+NOPLT

C-----
• 76 ILST=250
• GO TO 83

C-----
• 77 ILST=90

C-----
• GO TO 83

C-----
• 78 ILST=200
• GO TO 83

C-----
• 79 ILST=50
• GO TO 83

C-----
• 80 ILST=150
• GO TO 83

C-----
• 81 ILST=30

C-----
• INITIALIZE PLANT VARIABLES AND READ AN EMPLOYEE RECORD FROM DISK.
C-----

65 READ(INDX'ILST) LAST
WRITE(3.5) COMP, NDWK
5
1 'W/E '2(A2,''),A2//31' CHECK NO'7X'NAME'14X'AMOUNT')/)
T=0.
L=1
I=0
655 READ(NOPLT'L) NUM. NAME. NSS4N► NSTAS. NDUES. NWKMP 1►16A2.5X. PAY06
1	 NXMPF. NXMPS. NSEX. NRATE. YTD. ORTD. LYRHR. NCU. NCUDD. 
2	 NCHCK. NADWH. NSTCK. NINS. NMISC. NUA. NSTKD. ISUPP. INIT. 
3	 IPD, IFILL, GROSS, IVRA. IODRT. RGHR. OTHRS. BNHRS, RGERN. 
4	 OTERN. BNERN. OTHER. KO. HOLDY. VACA. SICK. CNET, IFICA, TAX. 
5	 LOCAL. ICU. IUA. IUD. IINS. ISTCK. IMISC 
C-----
CHECK PAID INDICATOR TO SEE IF CHECK WRITTEN.
C-----
IF(IPD .. 2) 650.651.650
C-----
PUT TOGETHER CHECK REGISTER INFORMATION.
C-----
651 T=T + CNET
I=I+ 1
GO TO (601.602.603) ;1
601 ID1=NCHCK
ID2=NUM
CALL MOVE(NAME.1.9*ID3+1)
REPORT=WHOLE(CNET + (CNET / ABS(CNET)) * 0.5) / 100. 
GO TO 650
602 ID4=NCHCK
ID5=NUM
CALL MOVE(NAME.1.9*ID6+1)
REPORT=WHOLE(CNET + (CNET / ABS(CNET)) * 0.5) / 100. 
GO TO 650
603 ID7=NCHCK
ID8=NUM
CALL MOVE(NAME.1.9*ID9+1)
REPORT=WHOLE(CNET + (CNET / ABS(CNET)) * 0.5) / 100. 
C-----
WRITE A LINE OF CHECK REGISTER FOR THREE EMPLOYEES.
C-----
WRITE(3.110) ID1. ID2. REPORT, ID7. ID8. REPORT, ID9. ID10. PAY06
1=0
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
C-----
PAGE 04

C----- HAVE WE PROCESSED THE LAST EMPLOYEE RECORD
PAY06

C----- YES - 657
PAY06
C----- NO - 655
PAY06

C----- IFIL = LAST1 657+655+637
PAY06

C----- IF THERE IS A PARTIAL LINE TO WRITE 16151, WRITE IT.
PAY06

C----- WRITE(3)(111) T
PAY06

C----- WRITE THE PLANT TOTAL
PAY06

C----- STOP
PAY06

CALL EXIT
PAY06

END
PAY06

VARIABLE ALLOCATIONS
PAY06

ICOL .0058  INVA .0058  MUNC .0058  LBO .0058  LBT .0058  LMC .0058  IN1 .0058  IN2 .0058  IN3 .0058  IN4 .0058
PAY06
INS .0058  IN6 .0058  FIBRE .0072  QRTD .004B  YTD .004A  TA .004B  YOE .004A  TotP .004D  TotB .004D
PAY06
TOTA+DC0  CMKX .004D  TGKS .004D  TNET .004D  TCA .004D  TGRO .004D  T1O0 .004D  T103 .004D  T108 .004D
PAY06

OTHERS=DOE  BNER=EDO  OTHER=DOE  HOLD=DOE  VACA .002A  SICK .002D  CNET .002D  RNE1 .002D  RNE2 .002D
PAY06
IDATE+101  ID9 .010A  ID6 .0111  ID5 .011C  ISUPP .0129  ITOT .0135  NAME .0130  NNAME .0140  NSAN .0143  COMP .0145
PAY06

TAX .0131  IC .0139  INPLT .0146  IBRO .0157  IWE1 .0156  IWE2 .0159  IC1 .0156  INDX .015B  ILS .015C  LAST .0152
PAY06

C----- NMA .015E  IN .015E  RNET1 .005C  RNET2 .005C  WNET3 .005C  IDATE .0101  103 .010A  106 .0113  109 .011C
PAY06

STATEMENT ALLOCATIONS
PAY06

ONE WORD INTEGERS
PAY06

EXTENDED PRECISION
PAY06

CORE REQUIREMENTS FOR PAY06
PAY06

FILE REQUIREMENTS FOR PAY06
PAY06

COMMON 0 VARIABLES 392 PROGRAM 668
PAY06

END OF COMPI LATION
PAY06
CHECK REGISTER

FACTORY PAYROLL THE CONTAINER CORP.
W/E 02-15-68

<table>
<thead>
<tr>
<th>CHECK NO</th>
<th>NAME</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ROBT H BADEN</td>
<td>86.08</td>
</tr>
<tr>
<td>2</td>
<td>JOHN A HORN</td>
<td>83.55</td>
</tr>
<tr>
<td>3</td>
<td>JOSEPH MONTANO</td>
<td>142.58</td>
</tr>
<tr>
<td>4</td>
<td>JOHN W CUSSEN</td>
<td>86.26</td>
</tr>
<tr>
<td>5</td>
<td>DONALD MILLER</td>
<td>129.33</td>
</tr>
<tr>
<td>6</td>
<td>A E TAYLOR</td>
<td>113.63</td>
</tr>
<tr>
<td>7</td>
<td>FRANK T DOLEN</td>
<td>81.53</td>
</tr>
<tr>
<td>8</td>
<td>DAVID A HUBBARD</td>
<td>88.48</td>
</tr>
<tr>
<td>9</td>
<td>ROST L SHORES</td>
<td>61.44</td>
</tr>
<tr>
<td>10</td>
<td>4L REYNOLDS</td>
<td>123.97</td>
</tr>
</tbody>
</table>

TOTAL   996.85

Input cards

Output on printer
<table>
<thead>
<tr>
<th>IBM 1130 MACHINE SETUP SHEET</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM NAME:</td>
</tr>
<tr>
<td>PROGRAM DESCRIPTION:</td>
</tr>
<tr>
<td>PRINTER</td>
</tr>
<tr>
<td>Standard</td>
</tr>
<tr>
<td>DISKS</td>
</tr>
<tr>
<td>CARTRIDGE ID:</td>
</tr>
<tr>
<td>SWITCH SETTINGS</td>
</tr>
<tr>
<td>None</td>
</tr>
<tr>
<td>INPUT CARDS</td>
</tr>
<tr>
<td>SOURCE OF INPUT:</td>
</tr>
<tr>
<td>DISPOSITION OF OUTPUT:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS</td>
</tr>
</tbody>
</table>
PAY09: 941 REPORT
## VARIABLES

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>P</td>
<td>30</td>
<td></td>
<td>0.000000</td>
<td>0.000000</td>
<td>Used to calculate overtime rate</td>
</tr>
<tr>
<td>FICA</td>
<td>R</td>
<td>30</td>
<td></td>
<td>0.000000</td>
<td>0.000000</td>
<td>FICA taxable wages</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>1</td>
<td>T</td>
<td>-</td>
<td>-</td>
<td>Used in DO loop</td>
</tr>
<tr>
<td>IC</td>
<td>I</td>
<td>1</td>
<td>T</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>ICOL</td>
<td>I</td>
<td>1</td>
<td>T</td>
<td>250</td>
<td>I</td>
<td>Record number in employee file set up by plant</td>
</tr>
<tr>
<td>IDATE</td>
<td>I</td>
<td>3,5,10</td>
<td></td>
<td>-</td>
<td>-</td>
<td>Pay date</td>
</tr>
<tr>
<td>IHD1</td>
<td>42</td>
<td>22,10</td>
<td></td>
<td>-</td>
<td>-</td>
<td>1st line of heading</td>
</tr>
<tr>
<td>IHD2</td>
<td>42</td>
<td>22,10</td>
<td></td>
<td>-</td>
<td>-</td>
<td>2nd line of heading</td>
</tr>
<tr>
<td>IHD3</td>
<td>42</td>
<td>22,10</td>
<td></td>
<td>-</td>
<td>-</td>
<td>3rd line of heading</td>
</tr>
<tr>
<td>IHD4</td>
<td>42</td>
<td>22,10</td>
<td></td>
<td>-</td>
<td>-</td>
<td>4th line of heading</td>
</tr>
<tr>
<td>IL</td>
<td>A</td>
<td>1</td>
<td>0,1</td>
<td>6</td>
<td></td>
<td>Carriage control</td>
</tr>
<tr>
<td>INDX</td>
<td>I</td>
<td>1</td>
<td>T</td>
<td>106</td>
<td>101</td>
<td>Index file number (plant no + 100)</td>
</tr>
<tr>
<td>INIT</td>
<td>I</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Union initiation fees</td>
</tr>
<tr>
<td>IN1</td>
<td>I</td>
<td>1</td>
<td>T</td>
<td>250</td>
<td>I</td>
<td>Record number in indexes to employee files</td>
</tr>
<tr>
<td>IN2</td>
<td>I</td>
<td>1</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN3</td>
<td>I</td>
<td>1</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN4</td>
<td>I</td>
<td>1</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN5</td>
<td>I</td>
<td>1</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IN6</td>
<td>I</td>
<td>1</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>Equivalent to IN1</td>
</tr>
<tr>
<td>IPAGE</td>
<td>I</td>
<td>1</td>
<td>0</td>
<td>20</td>
<td>1</td>
<td>Page number</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/TEMP</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPD</td>
<td>I/O</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>Indicates status of record in processing cycle</td>
</tr>
<tr>
<td>ISSAN</td>
<td>A/90</td>
<td>0</td>
<td>ALWAYS 9 DIGITS</td>
<td></td>
<td></td>
<td>Social Security</td>
</tr>
<tr>
<td>ISUPP</td>
<td>I/80</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>Supplemental sick pay</td>
</tr>
<tr>
<td>IWYA</td>
<td>I/T</td>
<td>250</td>
<td>1</td>
<td></td>
<td></td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LAST</td>
<td>I/T</td>
<td>XXX</td>
<td></td>
<td>0</td>
<td></td>
<td>Last record number in file</td>
</tr>
<tr>
<td>LBO</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LBT</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LINE</td>
<td>I/T</td>
<td>50</td>
<td></td>
<td>0</td>
<td></td>
<td>Line count</td>
</tr>
<tr>
<td>LMC</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>LST</td>
<td>I/T</td>
<td>250</td>
<td>250</td>
<td></td>
<td></td>
<td>Last record number in a file</td>
</tr>
<tr>
<td>LVRHP</td>
<td>I/O</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>This year's accumulation of hours worked for vacation pay</td>
</tr>
<tr>
<td>M1AR</td>
<td>I/I0</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td>Marital status-(1-single),(2-married)</td>
</tr>
<tr>
<td>MPCO</td>
<td>I/O</td>
<td>288</td>
<td></td>
<td>0</td>
<td></td>
<td>Number of employees reported per company</td>
</tr>
<tr>
<td>MPLY</td>
<td>I/O</td>
<td>41</td>
<td></td>
<td>0</td>
<td></td>
<td>Number of employees reported per page</td>
</tr>
<tr>
<td>MUCI</td>
<td>I/N</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td>Equivalent to ICOL</td>
</tr>
<tr>
<td>N</td>
<td>I/I</td>
<td>G</td>
<td>1</td>
<td></td>
<td></td>
<td>Plant number</td>
</tr>
<tr>
<td>NADWH</td>
<td>I/O</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>Additional withholding amount</td>
</tr>
<tr>
<td>NAME</td>
<td>A/90</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td>Dummy area to allocate space for name</td>
</tr>
<tr>
<td>NCNCK</td>
<td>I/O</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>Check number used for this employee</td>
</tr>
<tr>
<td>NCU</td>
<td>I/I</td>
<td>XX.XX</td>
<td></td>
<td>0</td>
<td>0</td>
<td>Credit union deduction</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic*
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT/OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCUDD</td>
<td>I, I</td>
<td>0</td>
<td>F</td>
<td>0</td>
<td>0</td>
<td>Monthly credit</td>
</tr>
<tr>
<td>NDUES</td>
<td>I, I</td>
<td>XXX</td>
<td>F</td>
<td>0</td>
<td>0</td>
<td>Union dues deduction</td>
</tr>
<tr>
<td>NINS</td>
<td>I, I</td>
<td>XXX</td>
<td>F</td>
<td>0</td>
<td>0</td>
<td>Insurance deduction</td>
</tr>
<tr>
<td>NHISC</td>
<td>I, I</td>
<td>0</td>
<td>F</td>
<td>0</td>
<td>0</td>
<td>Miscellaneous deductions</td>
</tr>
<tr>
<td>INRATE</td>
<td>I, I</td>
<td>3, 0, 125</td>
<td>F</td>
<td></td>
<td></td>
<td>Employee pay rate</td>
</tr>
<tr>
<td>NSEX</td>
<td>I, I</td>
<td>3</td>
<td>F</td>
<td>1</td>
<td></td>
<td>Sex-(1-female), (2-male), (3-trucker)</td>
</tr>
<tr>
<td>NSSAN</td>
<td>3</td>
<td>0, 3</td>
<td>F</td>
<td>0</td>
<td></td>
<td>Employee status-(1-union), (2-trucker)</td>
</tr>
<tr>
<td>NSTAS</td>
<td>I, I</td>
<td>0, 5</td>
<td>F</td>
<td>1</td>
<td></td>
<td>Employee status-(1-union), (2-trucker), (3-non-union, Full time), (4-non-union, part time), (5-terminated)</td>
</tr>
<tr>
<td>NSTCK</td>
<td>I, I</td>
<td>XXX</td>
<td>F</td>
<td>0</td>
<td></td>
<td>Stock deduction</td>
</tr>
<tr>
<td>NSTKDI</td>
<td>I, I</td>
<td>0</td>
<td>F</td>
<td>0</td>
<td>0</td>
<td>Monthly stock deductions</td>
</tr>
<tr>
<td>NUA</td>
<td>I, I</td>
<td>XXX</td>
<td>F</td>
<td>0</td>
<td></td>
<td>United Appeal deductions</td>
</tr>
<tr>
<td>NUM</td>
<td>I, I</td>
<td>XXX</td>
<td>F</td>
<td>1000</td>
<td></td>
<td>Clock number</td>
</tr>
<tr>
<td>NWKMP</td>
<td>I, I</td>
<td>0</td>
<td>F</td>
<td>0</td>
<td>0</td>
<td>Number of weeks employed</td>
</tr>
<tr>
<td>NWKPD</td>
<td>I, I</td>
<td>0</td>
<td>F</td>
<td>0</td>
<td>0</td>
<td>Number of weeks paid</td>
</tr>
<tr>
<td>NXMPE</td>
<td>I, I</td>
<td>17</td>
<td>F</td>
<td>0</td>
<td></td>
<td>Federal exemptions</td>
</tr>
<tr>
<td>NXMPS</td>
<td>I, I</td>
<td>17</td>
<td>F</td>
<td>0</td>
<td></td>
<td>State exemptions</td>
</tr>
<tr>
<td>QRTO</td>
<td>R</td>
<td>0</td>
<td>F</td>
<td>0</td>
<td></td>
<td>Quarterly-to-date Information (1) gross, (2) net, (3) FICA (4) loc. tax, (5) FICA wages, (6) sick pay</td>
</tr>
<tr>
<td>TOTA</td>
<td>R</td>
<td>0</td>
<td>F</td>
<td>0</td>
<td></td>
<td>Total FICA wages per page</td>
</tr>
<tr>
<td>TOTB</td>
<td>R</td>
<td>0</td>
<td>F</td>
<td>0</td>
<td></td>
<td>Total wages per page</td>
</tr>
<tr>
<td>TOTC</td>
<td>R</td>
<td>0</td>
<td>F</td>
<td>0</td>
<td></td>
<td>Total FICA wages per company</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE*</th>
<th>No. of Words</th>
<th>INPUT/TEMP OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTD</td>
<td>R 30</td>
<td>xxxxx</td>
<td></td>
<td>0.00</td>
<td></td>
<td>Total wages per company</td>
</tr>
<tr>
<td>YTD</td>
<td>R 42</td>
<td>xxxxx</td>
<td></td>
<td>0.00</td>
<td></td>
<td>Year-to-Date information: (1) gross, (2) FICA, (3) FICA wages, (4) sick pay, (5) spec A, (6) spec B, (7) loc. tax, (8) reg. hours, (9) overtime hours, (10) bonus hours, (11) reg. earn, (12) OT earn, (13) OT earn, (14) bonus earn.</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic
// FOR
+ 10CS(CARD+TYPEWRITER, 1132 PRINTER+DISK)
+ NAME RAY09
+ ONE WORD INTEGERS
+ EXTENDED PRECISION
+ LIST ALL
+ C----- JOB NAME -- PAYROLL SYSTEM - 941 REPORT
C----- JOB NUMBER -- PAY09
C----- PROGRAMMER -- C.R.KLICK
C----- DATE CODED -- 02/03/68
C----- DATE UPDATED --
C----- INPUT FILES --
FILE NAME NUMBER LENGTH RECORDS PER SECTOR PAY09
1. COLFP 1 160 250 2 PAY09
2. WVAPP 2 160 90 2 PAY09
3. MNCFP 3 160 200 2 PAY09
4. LBOFP 4 160 50 2 PAY09
5. LMCFP 5 160 150 2 PAY09
6. INDX1 101 1 250 320 PAY09
7. INDX2 102 1 90 320 PAY09
8. INDX3 103 1 200 320 PAY09
9. INDX4 104 1 50 320 PAY09
10. INDX5 105 1 150 320 PAY09
11. INDX6 106 1 30 320 PAY09
12. INDX7 107 1 10 320 PAY09
C----- OUTPUT FILES -- NONE
C----- ALLOCATE ARRAY STORAGE
DIMENSION IDATE(3), IHD1(22), IHD2(22), IHD3(22), IHD4(22),
ISSAN(9), ISUPP(13), NAME(9), NSSANN(3), ORTD(6), YTD(14)
C----- DEFINE THE FILES FOR THIS PROGRAM AS DESCRIBED ABOVE, AND
C----- EQUIVALENCE THE VARIABLES FOR THE NEXT RECORD NUMBER.
DEFINE FILE (250+160+U,ICOL), (190+160+U,IWVA),
(200+160+U,MUNC), (50+160+U,LBO),
(150+160+U,LBT), (30+160+U,LMC),
(250+1+U,IN1), (102+1+U,IN2), (103+1+U,IN3),
(150+1+U,IN4), (106+30+1+U,IN6),
(1+U,IN1), (1+U,IN2), (1+U,IN3), (1+U,IN4), (1+U,IN5), (1+U,IN6)
C----- INITIALIZE VARIABLES AND READ PLANT NO., DATE, AND PAGE NO.
C----- IN=16448
C-----
1000 READ(2,1) N, IDATE, IPAGE
1 FORMAT(1x,4I2)
C-----
C----- IS THIS THE LAST PLANT?
C----- YES = 99
C----- NO = 110
C----- IF(IN) 99,99,100
C----- 100 IF(IN = 6) 110,10,99
C----- C----- READ THE PLANT NAME AND ADDRESS, AND INITIALIZE THE REMAINING
C----- VARIABLES, AND WRITE PLANT INFORMATION ON TOP OF FIRST PAGE.
C----- C----- 110 READ(2,2) IHD1, IHD2, IHD3, IHD4
C----- 2 FORMAT(2A2)
C----- C----- MPCI=0
C----- MPLY=0
C----- TOTA=0
C----- TOTA0=0
C----- TOTA=0
C----- TOTA=0
C----- LINE=0
C----- WRITE(3,3) IL, IHD1, IDATE, IPAGE, IHD2, IHD3, IHD4
3 FORMAT(12X,2X,2A2,2X,2(12X,''),12X,10X,2(13X,2A2))
C----- WRITE(3,8)
8 FORMAT('1')
C----- IL=-3776
C----- IPAGE=IPAGE + 1
C----- INDX=N + 100
C----- GO TO (131,132,133,134,135,136) N
C----- 131 LST=250
C----- GO TO 140
C----- 132 LST=90
C----- GO TO 140
C----- 133 LST=200
C----- GO TO 140
C----- 134 LST=50
C----- GO TO 140
C----- 135 LST=150
C----- GO TO 140
C----- 136 LST=30
C----- C----- GET THE NUMBER OF EMPLOYEES
C----- C----- 140 READ(INDX,LST) LAST

PAGE 02

SECTION 35

SUBSESSIONS 20

PAGE 139
C \ READ AN EMPLOYEE RECORD FROM DISK AND DECIDE IF IT COUNTS.  
\ DO 275 I=1,LAST  
\ READIN II) NUM, NAME, NSSAN, NSTAS, NDUES, NWKMP, NWKPD, MAR,  
\ 1 NCHCK, NADWH, NSTCK, NINS, NMISC, NUA, NSTKD, ISUPP, INIT, IPD  
\ 2 WHOLC, TOTA, TOTB, TOTA=WHOLE(TOTA + (TOTA / ABS(TOTA)) * 0.5) / 100.  
\ TOTB=WHOLE(TOTB + (TOTB / ABS(TOTB)) * 0.5) / 100.  
\ IF(RQTD(1)) 150 275 150  
\ THIS ROUTINE CONTROLS THE PAGE FORMAT, IF 40 LINES HAVE BEEN  
\ PRINTED PUT HEADINGS AT TOP OF NEXT PAGE, OTHERWISE DO NOTHING.  
\ IF(LINE = 40) 170 170 160  
\ MPCO=MPCO + MPLY  
\ TOTC=TOTC + IOTA  
\ TOTD=TOTD + TOTB  
\ TOTA=WHOLE(TOTA + (TOTA / ABS(TOTA)) * 0.5) / 100.  
\ TOTB=WHOLE(TOTB + (TOTB / ABS(TOTB)) * 0.5) / 100.  
\ WRITE TOTALS AT THE BOTTOM OF THE PAGE.  
\ WRITE(3,5) MPLY, MPLY, IOTA, TOTB  
\ 5 FORMAT(111,30X,I2.8X.I217X.F9.2.4X.F9.2)  
\ MPLY=0  
\ NEXT PAGE  
\ WRITE(3,4) IHFl, IDATE, IPAGE, IH01, IH02, IH03, IH04  
\ 4 FORMAT(1' '22A2,2X,22A2,2X,22A2,2X,22A2,2A2)  
\ WRITE(3,8)  
\ LINE=0  
\ IPAGE=IPAGE + 1  
\ TOTA=0  
\ TOTB=0  
\ NEXT PAGE  
\ ADD EMPLOYEE INFORMATION TO TOTAL AND SETUP DETAIL LINES.  
\ 170 A=NSSAN(1)  
\ CALL PUT(ISSAN+13X*A + 10*5+1)  
\ A=NSSAN(2)  
\ CALL PUT(ISSAN+45X*A + 10*5+1)  
\ A=NSSAN(3)  
\ CALL PUT(ISSAN+9X*A + 10*5+1)
THE PROGRAM COMES THRU HERE WHEN THE LAST PLANT HAS BEEN PROCESSED.

CALL EXIT

END

CORE REQUIREMENTS FOR PAY09

COMMON 0 VARIABLES 320 PROGRAM 826

END OF COMPILATION
// JOB
// JOB PAY09 2
*FILES(1*COLFP),(2*WVAFP),(3*MCNFP),(4*LBGFP),(5*LBTFP),(6*LMCFP),
*FILES(101*INDX1),(102*INDX2),(103*INDX3),(104*INDX4),(105*INDX5),(106*INDX6)

103916601
XYZ MANUFACTURING COMPANY
1642 EAST MIDDLETOWN STREET
ANYTOWN, SOMESTATE 99999
013-32-3060

Input cards
**CONTINUATION SHEET FOR SCHEDULE A OF FORMS 941, 941-E, 941-SS, OR 943**

**THE CONTAINER COMPANY**

1642 EAST MIDDLETOWN STREET
COLUMBUS, WASHINGTON 99999
013-32-3060

**Type or print in W-2, make employer, identification number, name, and address exactly as shown on the return.**

**DATE ETTED** 3-31-68

**If this form is used as a continuation sheet for Form 943, Employer's Annual Tax Return for Agricultural Employees, please check here.**

**READ INSTRUCTIONS CAREFULLY**

**Attach only original continuation sheets to your tax return. Do not send a carbon copy to the U.S. District Director of Internal Revenue.**

---

<table>
<thead>
<tr>
<th>EMPLOYER'S SOCIAL SECURITY ACCOUNT NUMBER</th>
<th>NAME OF EMPLOYEE</th>
<th>TAXABLE F.I.C.A. WAGES</th>
<th>TAXABLE TIPS REPORTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>013 32 3060</td>
<td>ROBERT B BADEN</td>
<td>1831.01</td>
<td>1831.01</td>
</tr>
<tr>
<td>083 28 4339</td>
<td>JOHN A HORN</td>
<td>2202.84</td>
<td>2202.84</td>
</tr>
<tr>
<td>712 98 2119</td>
<td>ROBT L SHORES</td>
<td>1906.65</td>
<td>1906.65</td>
</tr>
<tr>
<td>032 24 4376</td>
<td>JOHN W CUSSEN</td>
<td>2286.25</td>
<td>2286.25</td>
</tr>
<tr>
<td>614 63 8734</td>
<td>JOSEPH MONTANO</td>
<td>3176.73</td>
<td>3176.73</td>
</tr>
<tr>
<td>541 03 2308</td>
<td>DONALD MILLER</td>
<td>1346.00</td>
<td>1346.00</td>
</tr>
<tr>
<td>213 71 0014</td>
<td>A E TAYLOR</td>
<td>2233.03</td>
<td>2233.03</td>
</tr>
<tr>
<td>782 92 7112</td>
<td>DAVID A HUBBARD</td>
<td>1923.58</td>
<td>1923.58</td>
</tr>
<tr>
<td>194 51 1234</td>
<td>FRANK T DOLEN</td>
<td>1475.89</td>
<td>1475.89</td>
</tr>
<tr>
<td>822 44 5678</td>
<td>AL REYNOLDS</td>
<td>3142.25</td>
<td>3142.25</td>
</tr>
</tbody>
</table>

**TOTALS FOR THIS PAGE**

NUMBER OF EMPLOYEES: 10
21520.23
21532.23

**FEDERAL COPY**

---

**XYZ MANUFACTURING COMPANY**

10 21520.23 21532.23

---

140
IBM 1130 MACHINE SETUP SHEET

PROGRAM NAME: 941 REPORT
PROGRAM NUMBER: PAY09

PROGRAM DESCRIPTION: APPROXIMATE RUNNING TIME:

<table>
<thead>
<tr>
<th>PRINTER</th>
<th>TYPE OF PAPER</th>
<th>NO. OF COPIES</th>
<th>CARRIAGE TAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>941 FORMS</td>
<td></td>
<td>941 TAPE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISKS</th>
<th>DRIVE NUMBER:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CARTRIDGE ID: PAYROLL</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>SWITCH SETTINGS</th>
<th>SWITCH</th>
<th>SWITCH</th>
<th>SWITCH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UP</td>
<td>DOWN</td>
<td>UP</td>
</tr>
</tbody>
</table>

INPUT CARDS

For one plant:
- MORE PLANTS
- PLANT ACCOUNT NO. CARD
- PLANT CITY-STATE CARD
- PLANT ADDRESS CARD
- PLANT NAME CARD
- PLANT HEADER CARD
- 11 XEQ PAY09
- 11 JOB

SOURCE OF INPUT:
1. Plant information cards from file E
2. Payroll disk from storage

DISPOSITION OF OUTPUT:
1. 941 report to government
2. Disk is returned to storage
3. Plant information cards to file E

FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS
PAYROLL SYSTEM

Operation Manual
**CONTENTS**

Payroll Application ................................................................. 1  
Job Description ................................................................. 1  
System Flowchart ................................................................. 1  
  Narrative ................................................................. 1  
  Payroll File Create (PAY01, PAY02, PAY16) ......................... 2  
  Payroll File Changes (PAY03, PAY16) ............................ 3  
  Payroll Calculations and Register (PAY04, PAY16) .......... 4  
  Print Payroll Checks (PAY05, PAY06) ............................... 5  
  Payroll Check Voiding (PAY011) ....................................... 6  
  Payroll Deduction Registers (PAY12 thru PAY15) ............. 7  
  Payroll File Audit, 941, and Tax (PAY07, PAY09, PAY10) ... 8  
  Print W-2 Reports (PAYnn) ........................................... 9  
  Error Detection and Correction (PAY09) ......................... 10  
Payroll Programs ................................................................. 11  
  PAY01: Payroll File Create ........................................... 11  
    Accounting Controls .............................................. 11  
    Error Recovery Sheet .......................................... 12  
    Machine Setup Sheet ........................................... 13  
  PAY02: Add Names to the File ......................................... 14  
    Accounting Controls .............................................. 14  
    Error Recovery Sheet .......................................... 15  
    Machine Setup Sheet ........................................... 17  
  PAY03: Changes to the File ........................................... 18  
    Accounting Controls .............................................. 18  
    Error Recovery Sheet .......................................... 19  
    Machine Setup Sheet ........................................... 25  
  PAY04: Calculations and Payroll Register ......................... 26  
    Accounting Controls .............................................. 26  
    Error Recovery Sheet .......................................... 27  
    Machine Setup Sheet ........................................... 37  
  PAY05: Check Writing ................................................... 38  
    Accounting Controls .............................................. 38  
    Error Recovery Sheet .......................................... 39  
    Machine Setup Sheet ........................................... 50  
  PAY06: Check Register ................................................... 51  
    Accounting Controls .............................................. 51  
    Error Recovery Sheet .......................................... 52  
    Machine Setup Sheet ........................................... 53  
  PAY09: 941 Report ......................................................... 54  
    Accounting Controls .............................................. 54  
    Error Recovery Sheet .......................................... 55  
    Machine Setup Sheet ........................................... 56  

PAYROLL APPLICATION

JOB DESCRIPTION

The Payroll System is composed of 16 different runs. From the source documents, produced at the six plant sites, cards are punched. These cards are used to store the payroll information on the disk cartridge.

At this point the system uses cards only for transition between jobs. The input data, employee records, is read from the disk and updated before being written back. This gives a highly flexible system, in which I/O, because of the disk, is very fast.

The system produces the following reports:

- Checks and check stubs
- Check register
- Payroll register
- Deduction registers for
  1. Union dues
  2. Credit union
  3. Stock
- 941 quarterly report

SYSTEM FLOWCHART

Narrative

The system consists of 16 programs.

The Files Creation program is first. Data decks are keypunched for each individual, in sets, by plant. The data is edited and, when correct, loaded on the disk by PAY01. Three files are created: a master file, an index file, and a plant information file. A second data deck with employee clock number and name is loaded onto the master file by PAY02.

Changes to the disk information are made by PAY03. Documents, received from personnel departments at the individual plants, are checked, summarized, keypunched, and verified. Time sheets, submitted by the plant payroll departments, are keypunched and verified. All of these cards are processed by PAY16, which edits and generates control totals. PAY04 then processes these cards, performing all payroll calculations. Cards are read, pay computed, disk files updated, and cards extended with current pay figures. After all cards are processed, a payroll register is printed.

Checks are printed by PAY05. A header card is read and the checks are printed from the disk file. PAY06 lists the check register from the disk file. In the event of an error in computing pay, PAY11 provides the means of voiding checks. The extended time cards from PAY04 are read in and the affected employee records are reset. The above are weekly runs.

At month end, registers are prepared showing each individual's deductions for the month:

PAY13 writes union dues register.
PAY14 writes credit union register.
PAY15 writes stock deductions register.
PAY12 resets charity deductions code.

At the end of the quarter and at the end of the year PAY07 and PAY08 are used to balance the disk files to control totals.

PAY09 produces the 941 tax report.
PAY10 produces a tax worksheet used to determine tax liability.

At the present time the program for W-2 reports has not been written.
Weekly Time Sheets

Totals on Adding Machine

Keypunch & Key-Verify Details

Keypunch & Key-Verify Control Totals

Details

Zero Balance Totals

PAY 16 INPUT EDIT

Control Totals

Balance to Totals & Correct as Necessary

Out of Balance

PAY 04 CALCULATION

Balance to Totals; If Incorrect, Go to E

Distribute Payroll Register

Disk Payroll File

Details

Control Totals

File D
Control
Totals
Burst, Sign
and Distribute
Paychecks
and Stubs
Balance to
Totals; If
Incorrect,
Go to E
Distribute
Check
Register 1
Disk
Payroll
File
(Control
Totals
File)

Calculated
Payroll
CHECKS

•
Balance to
Incorrect,
Go to D

Only When
Balance

Total on
Adding
Machine

TAPE

Section 35
Subsections 20
Page 09
Only When Totals Do Not Balance

- Destroy Checks
- Disk Payroll File
- Control Totals

PAY 11 VOID CHECKS

- Disk Payroll File
- Control Totals
- Details

File D

D

Details

To B

To C

File D
File Disk Payroll File

Totals

PAY.07 AUDIT FILE BY COMPANY

Enter Plant Number

Balance to Totals; If Incorrect, Go to E

Disk Payroll File

Calculated Control Totals

PAY.09 941 REPORT

941 Report

Balance to Totals; If Incorrect, Go to E

Distribute 941 Report

Disk Payroll File

Plant Numbers

PAY.10 TAX WORKSHEET

Balance to Totals; If Incorrect, Go to E

Distribute Tax Worksheet

Disk Payroll File

Plant Numbers

File E
From Storage

Disk Payroll File

W-2 Reports

PAYon W-2 REPORTS

Balance to Totals; If Incorrect, Go to E

Distribute W-2 Reports

Disk Payroll File

Store

File E

File E

Plant Numbers

E Tape

General Ledger

Totals on Adding Machine

Disk General Payroll Plant Ledger File Numbers

W-2 Payroll Plant Reports File Numbers

Store
Individual Payroll Record

PAY 08 INQUIRY

Balance to Totals; If Correct, Go to E

Determine Change Required

Use PAY 16 & PAY 03 to Change the Disk Payroll Record

Does this correct original error? If not, Go to E

Return to Print Where Error Occurred Only when entire original error has been corrected
PAYROLL PROGRAMS

PAY01: PAYROLL FILE CREATE

Accounting Controls

Balance total gross ($) and total tax withheld YTD ($) from the preceding PAY16 to the general ledger.
IBM 1130 ERROR RECOVERY SHEET

<table>
<thead>
<tr>
<th>JOB</th>
<th>PROGRAM NAME</th>
<th>PROGRAMMER NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payroll System</td>
<td>PAY01</td>
<td>CRKlick</td>
</tr>
</tbody>
</table>

MESSAGE TYPED:  
PAUSE - DISPLAYED IN ACCUM:  

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT:  

DESCRIPTION OF WHAT IS WRONG:  

PROBABLE CAUSE:  

RECOVERY PROCEDURES:  

COMMENTS:  

There are no messages or pauses.

SCORESHEET

<table>
<thead>
<tr>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### IBM 1130 MACHINE SETUP SHEET

<table>
<thead>
<tr>
<th>PROGRAM NAME:</th>
<th>File Create</th>
<th>PROGRAM NUMBER:</th>
<th>PAY01</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM DESCRIPTION:</td>
<td></td>
<td>APPROXIMATE RUNNING TIME:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRINTER</th>
<th>TYPE OF PAPER</th>
<th>NO. OF COPIES</th>
<th>CARRIAGE TAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>Standard</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISKS</th>
<th>DRIVE NUMBER:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARTRIDGE ID:</td>
<td>Payroll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWITCH SETTINGS</th>
<th>SWITCH</th>
<th>UP</th>
<th>DOWN</th>
<th>SWITCH</th>
<th>UP</th>
<th>DOWN</th>
<th>SWITCH</th>
<th>UP</th>
<th>DOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INPUT CARDS</th>
</tr>
</thead>
</table>

```
   a
   For one plant
   DETAIL CARDS
   /I XEQ PAY01
   /I JOB
```

### SOURCE OF INPUT:
1. Card input from a successful PAY02 edit run.
2. Disk must be payroll disk with areas for each plant allocated.

### DISPOSITION OF OUTPUT:
1. Detail cards are filed in File A.
2. Disk to be used in PAY02, which should be run next.

FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS
PAY02: ADD NAMES TO THE FILE

Accounting Controls

None
IBM 1130 ERROR RECOVERY SHEET

JOB Payroll System
PROGRAM NAME PAY02
PROGRAMMER NAME C.R. Eich

MESSAGE TYPED: 
CLOCK NO XXXX NOT IN FILE

PAUSE - DISPLAYED IN ACCUM: 
NONE

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT 100

DESCRIPTION OF WHAT IS WRONG:
The clock number in the input card is not in the index of clock numbers.

PROBABLE CAUSE:
1. The employee's record was not loaded.
2. The clock number in the input card is incorrect.

RECOVERY PROCEDURES: 
At the end of the run, remove the card(s) so noted from the data deck and check the clock number(s) with personnel records. If the clock number is correct, load the employee record on the file. If the card is incorrect, repunch the card and rerun.

COMMENTS:

SCORESHEET

| DATE | | | | | |
|------| | | | | |

INITIALS

15
IBM 1130 ERROR RECOVERY SHEET

JOB Payroll System
PROGRAM NAME PAY02
PROGRAMMER NAME CR. Kick

MESSAGE TYPED:
CLOCK NO. XXXX IN
FILE DOES NOT AGREE
WITH CLOCK NO. XXXX
IN CARD

PAUSE - DISPLAYED IN ACCUM:
NONE

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT 100

DESCRIPTION OF WHAT IS WRONG:
The clock number on the disk file is not
the same as the clock number in the index and
in the input card.

PROBABLE CAUSE:
Disk data has been altered.

RECOVERY PROCEDURES:
Immediately report the occurrence of
this to your supervisor.

COMMENTS:
Because the specific record in error
looks identification, this message has little
value. However, this indicates that disk data
has probably been destroyed.

SCORESHEET

<table>
<thead>
<tr>
<th>DATE</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**IBM 1130 MACHINE SETUP SHEET**

<table>
<thead>
<tr>
<th>PROGRAM NAME:</th>
<th>Add names to the file</th>
<th>PROGRAM NUMBER: PAYO2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM DESCRIPTION:</td>
<td></td>
<td>APPROXIMATE RUNNING TIME:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRINTER</th>
<th>TYPE OF PAPER</th>
<th>NO. OF COPIES</th>
<th>CARRIAGE TAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard</td>
<td>1</td>
<td>Standard</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISKS</th>
<th>DRIVE NUMBER:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CARTRIDGE ID:</td>
<td>Payroll</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWITCH SETTINGS</th>
<th>SWITCH</th>
<th>NONE</th>
<th>SWITCH</th>
<th>NONE</th>
<th>SWITCH</th>
<th>NONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP</td>
<td>UP</td>
<td></td>
<td>UP</td>
<td></td>
<td>UP</td>
<td></td>
</tr>
<tr>
<td>DOWN</td>
<td></td>
<td></td>
<td>DOWN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**INPUT CARDS**

For one plant

NAME & CLOCK NO. CARDS

11 XEQ PAYO2

11 JOB

**SOURCE OF INPUT:**
1. Card input for a successful PAYO2 edit run.
2. Disk must be payroll disk from PAYO1.

**DISPOSITION OF OUTPUT:**
1. Name and Clock No. cards are filed in file 8.
2. Disk to be used in PAYO3, which should be run next.

**FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS**
PAY03: CHANGES TO THE FILE

Accounting Controls

Hash totals of clock numbers, change codes and new fields from preceding PAY16 should balance to control totals prepared manually.
IBM 1130 ERROR RECOVERY SHEET

JOB: Payroll System
PROGRAM NAME: PAY03
PROGRAMMER NAME: C.R. Klick

MESSAGE TYPED:
PLANT NOS. DISAGREE
FOR CLOCK NUMBER
XXXX

PAUSE - DISPLAYED IN ACCUM:
NONE

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT 100

DESCRIPTION OF WHAT IS WRONG:
First digit of clock number
does not agree with the plant number in the
header card.

PROBABLE CAUSE:
1. Data for one plant was included with
data of another plant.
or
2. The deck is not setup correctly

RECOVERY PROCEDURES:
The card in error will be selected
to the alternate stacker. Remove the card
and compare it to the source document.
Correct it and rerun.

COMMENTS:
This error may result in a system F003
error. Reexecute the program starting with
the plant and card that was being processed
when the error occurred. If the error occurs a
second time, notify your supervisor.

SCORESHEET

<table>
<thead>
<tr>
<th>DATE</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# IBM 1131 ERROR RECOVERY SHEET

**JOB** | Payroll System  
**PROGRAM NAME** | PAY03  
**PROGRAMMER NAME** | CRES/XXK  

**MESSAGE TYPED:**

| INVALID CHANGE CODE FOR XXXX |

**PAUSE - DISPLAYED IN ACCUM:**

| NONE |

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT:**

| 100 |

**DESCRIPTION OF WHAT IS WRONG:**

The change code is not within the valid range (1-46).

**PROBABLE CAUSE:**

Keypunch error

**RECOVERY PROCEDURES:**

Return card and document to keypunch operator.

**COMMENTS:**


**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
</table>

20
IBM 1130 ERROR RECOVERY SHEET

**JOB**: Payroll System  
**PROGRAM NAME**: PAY03  
**PROGRAMMER NAME**: C.R. Klick

**MESSAGE TYPED:**  
**CLOCK NO XXXX NOT IN FILE**

**PAUSE – DISPLAYED IN ACCUM:**  
**NONE**

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT**:  
**100**

**DESCRIPTION OF WHAT IS WRONG:**  
The input card clock number is not in the index.

**PROBABLE CAUSE:**  
1. The employee record has not been loaded.
   or
2. The clock number on the card is punched incorrectly.

**RECOVERY PROCEDURES:**  
The card is stacker selected. Remove the card and check the clock number with personnel records. If the clock number is correct load the employee's record on the file. Otherwise, correct the card and rerun.

**COMMENTS:**

**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM 1130 ERROR RECOVERY SHEET</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td></td>
</tr>
<tr>
<td>JOB <strong>Payroll System</strong></td>
<td></td>
</tr>
<tr>
<td>PROGRAM NAME <strong>PAY03</strong></td>
<td></td>
</tr>
<tr>
<td>PROGRAMMER NAME <strong>C.R. Kick</strong></td>
<td></td>
</tr>
</tbody>
</table>

**MESSAGE TYPED:**
CLOCK NUMBERS DO NOT AGREE FOR XXXX.

**PAUSE - DISPLAYED IN ACCUM:**
NONE

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT:** 100

**DESCRIPTION OF WHAT IS WRONG:**
The clock number in the disk file is not the same as the clock number in the index and in the input card.

**PROBABLE CAUSE:**
Disk data has been altered.

**RECOVERY PROCEDURES:**
Card is stacker-selected.
Immediately report the occurrence of this error to your supervisor.

**COMMENTS:**
Disk data has probably been destroyed.

**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**IBM 1130 ERROR RECOVERY SHEET**

**JOB** Payroll System

**PROGRAM NAME** Pay03

**PROGRAMMER NAME** C. R. C. lick

**MESSAGE TYPED:**

ENTER SSAN FOR XXXX

**PAUSE - DISPLAYED IN ACCUM:**

![NONE]

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT**

Keyboard select

**DESCRIPTION OF WHAT IS WRONG:**

Machine is waiting for input from keyboard

**PROBABLE CAUSE:**

The program has called for input.

**RECOVERY PROCEDURES:**

Enter the Social Security number for the indicated employee.

**COMMENTS:**


## SCORESHEET

<table>
<thead>
<tr>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IBM 1130 ERROR RECOVERY SHEET

<table>
<thead>
<tr>
<th>JOB</th>
<th>Payroll System</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM NAME</td>
<td>PAY03</td>
</tr>
<tr>
<td>PROGRAMMER NAME</td>
<td>C.R. Klick</td>
</tr>
</tbody>
</table>

MESSAGE TYPED:  
CLOCK NUMBER XXXX IS DUPLICATED.

PAUSE - DISPLAYED IN ACCUM:  
NONE

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT 100

DESCRIPTION OF WHAT IS WRONG:  
The clock number in the input card for a new employee is already on the disk file.

PROBABLE CAUSE:  
1. The card has been entered a second time.
   or
2. The clock number in the input card is incorrect.

RECOVERY PROCEDURES:  
The card is stacker-selected. Check the clock number with personnel records. If it is incorrect, repunch and rerun. If it is correct, use the inquiry program, PAY08, to print the file record and determine whether or not it is correct.

COMMENTS:  

SCORESHEET

<table>
<thead>
<tr>
<th>DATE</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## IBM 1130 MACHINE SETUP SHEET

### PROGRAM NAME:
Changes to the file

### PROGRAM NUMBER:
PAY03

### PROGRAM DESCRIPTION:

### TYPE OF PAPER | NO. OF COPIES | CARRIAGE TAPE
--- | --- | ---
Standard | 1 | Standard

### PRINTER

<table>
<thead>
<tr>
<th>TYPE</th>
<th>NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>1</td>
</tr>
</tbody>
</table>

### DISKS

<table>
<thead>
<tr>
<th>DRIVE NUMBER</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARTRIDGE ID</td>
<td>Payroll</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SWITCH SETTINGS

<table>
<thead>
<tr>
<th>SWITCH</th>
<th>UP</th>
<th>DOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

### SOURCE OF INPUT:
1. Card input from a successful PAY16 edit run.
2. Disk must be payroll disk from files.

### DISPOSITION OF OUTPUT:
1. Change cards are filed in file C.
2. Disk is returned to storage for use with PAY04.

---

FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS
PAY04: CALCULATIONS AND PAYROLL REGISTER

Accounting Controls

Machine totals (regular hours, OT hours, bonus hours, special earnings) must be balanced to the control totals from the preceding PAY16 run. Information is found on console printer for this zero-balance check.
IBM 1130 ERROR RECOVERY SHEET

JOB Payroll System
PROGRAM NAME PAY04
PROGRAMMER NAME C. P. Klick

MESSAGE TYPED:
CHECK CC 1 AND 80 ON FIRST CARD

PAUSE - DISPLAYED IN ACCUM:
0 0 0 1

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT 99999

DESCRIPTION OF WHAT IS WRONG:
1. Card column 1 has an invalid plant number,
or
2. Card column 80 is not zero.

PROBABLE CAUSE:
Either a blank card or a date card has been placed in front of the deck.

RECOVERY PROCEDURES:
Clear the card reader (NPRO).
Place a proper header card at the beginning of the deck. Ready the card reader and press Program Start on the console.

COMMENTS:

SCORESHEET
DATE
INITIALS
IBM 1130 ERROR RECOVERY SHEET

JOB  Payroll System
PROGRAM NAME  PAY04
PROGRAMMER NAME  C. R. Klick

MESSAGE TYPED:

PAUSE - DISPLAYED IN ACCUM:

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT  No

DESCRIPTION OF WHAT IS WRONG:
Operator option to change constants

PROBABLE CAUSE:
Program allows for this possibility.

RECOVERY PROCEDURES:
Follow the instructions printed.

SCORESHEET

DATE
INITIALS
<table>
<thead>
<tr>
<th>JOB</th>
<th>Payroll System</th>
<th>PROGRAM NAME</th>
<th>PAY04</th>
<th>PROGRAMMER NAME</th>
<th>C.R. Klick</th>
</tr>
</thead>
</table>

**MESSAGE TYPED:**

CHECK CARD WITH CLOCK NUMBER XXX

**PAUSE - DISPLAYED IN ACCUM:**

0 1 0 0

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT:**

90

**DESCRIPTION OF WHAT IS WRONG:**

1. The first digit of the clock number does not agree with the plant number in the header card.

   or

2. Card column 90 is invalid.

**PROBABLE CAUSE:**

1. The data for one plant is included with the data for another plant.

   or

2. Card deck is not set up correctly.

**RECOVERY PROCEDURES:**

Clear the card reader. The first card to card is the card in error. Correct the card, if necessary, or remove it from the deck. Reload and ready the card reader. Press Program Start on the console.

**COMMENTS:**

The program will not continue until the card read is correct.

**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IBM 1130 ERROR RECOVERY SHEET

JOB: Payroll System
PROGRAM NAME: PAYO4
PROGRAMMER NAME: C.R. Klick

MESSAGE TYPED: 

PAUSE - DISPLAYED IN ACCUM:

CLOCK NO. XXXX IS NOT IN THE FILE

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT 120

DESCRIPTION OF WHAT IS WRONG:

Input record clock number is not on the disk.

PROBABLE CAUSE:
1. Employee record has not been loaded.
   or
2. Input number is incorrect.

RECOVERY PROCEDURES:
The card is stagger-selected. Check the clock number with personnel records. If the number is correct, load the employee's record. If it is incorrect, repunch and rerun.

COMMENTS:

SCORESHEET

DATE
INITIALS
**IBM 1130 ERROR RECOVERY SHEET**

**JOB** Payroll System  
**PROGRAM NAME** PAY04  
**PROGRAMMER NAME** C. R. Klick

**MESSAGE TYPED:**  
FILE NO. XXXX AND INDEX NO. XXXX DO NOT AGREE

**PAUSE – DISPLAYED IN ACCUM:**  
NONE

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT**  
120

**DESCRIPTION OF WHAT IS WRONG:**  
The clock number in the input card does not agree with the clock number in the employee record.

**PROBABLE CAUSE:**  
Disk data has been altered.

**RECOVERY PROCEDURES:**  
Card is stacker-selected.  
Immediately report this occurrence to your supervisor.

**COMMENTS:**  
Disk data has probably been destroyed.

**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**IBM 1130 ERROR RECOVERY SHEET**

**JOB** Payroll System  
**PROGRAM NAME** PAY04  
**PROGRAMMER NAME** C.R. Klick

**MESSAGE TYPED:**  
NET OF XXXXXX.  
FOR CLOCK NO XXXX

**PAUSE - DISPLAYED IN ACCUM:**  
NONE

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT 120**

**DESCRIPTION OF WHAT IS WRONG:**  
Net amount of check exceeds limit.

**PROBABLE CAUSE:**  
1. Limit set too low  
or  
2. Erroneous data in employee record

**RECOVERY PROCEDURES:**  
1. Change limit and rerun.  
or  
2. Correct employee record and rerun.

**COMMENTS:**

**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
</table>

32
**IBM 1130 ERROR RECOVERY SHEET**

**JOB** Payroll System  
**PROGRAM NAME** PAY04  
**PROGRAMMER NAME** C. R. Klick

<table>
<thead>
<tr>
<th>MESSAGE TYPED:</th>
<th>PAUSE - DISPLAYED IN ACCUM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT TOTALS</td>
<td>NONE</td>
</tr>
<tr>
<td>XX/0/0/r</td>
<td></td>
</tr>
<tr>
<td>in 59</td>
<td></td>
</tr>
</tbody>
</table>

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT** Next in sequence

**DESCRIPTION OF WHAT IS WRONG:** Nothing

**PROBABLE CAUSE:** End-of-job routine is printing out control totals from header card.

**RECOVERY PROCEDURES:**

**COMMENTS:**

**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IBM 1130 ERROR RECOVERY SHEET

**JOB:** Payroll System

**PROGRAM NAME:** PAY04

**PROGRAMMER NAME:** C.R. Klick

**MESSAGE TYPED:**

<table>
<thead>
<tr>
<th>PROCESSED TOTALS</th>
<th>PAUSE - DISPLAYED IN ACCUM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXXXXXXX. XXXXXXXX.</td>
<td>N O N E</td>
</tr>
<tr>
<td>XXXXXXXX.</td>
<td></td>
</tr>
</tbody>
</table>

**AFTER PAUSE, CONTROL TRANSfers TO STATEMENT:** NEXT in sequence

**DESCRIPTION OF WHAT IS WRONG:** Nothing

**PROBABLE CAUSE:** End-of-job routine is printing out accumulated control totals.

**RECOVERY PROCEDURES:**

**COMMENTS:**

**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
IBM 1130 ERROR RECOVERY SHEET

**JOB**  Payroll System

**PROGRAM NAME**  PAY04

**PROGRAMMER NAME**  C.R. Klick

---

**MESSAGE TYPED:**

- **ERROR TOTALS**
  - XXXXXX XXXXXX
  - XXXXXX XXXXXX

**PAUSE - DISPLAYED IN ACCUM:**

- NONE

---

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT:**

- Next in sequence

---

**DESCRIPTION OF WHAT IS WRONG:**

- Nothing

---

**PROBABLE CAUSE:**

- End-of-job routine is printing out accumulated totals of erroneous records.

---

**RECOVERY PROCEDURES:**

- Error figures must be accounted for and corrections made as necessary.

---

**COMMENTS:**

---

**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IBM 1130 ERROR RECOVERY SHEET

JOB Payroll System  PROGRAM NAME PAY04
PROGRAMMER NAME C.R. Klick

MESSAGE TYPED: THE DIFFERENCES x x x x x x x x x x x x x x
PAUSE - DISPLAYED IN ACCUM: NONE

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT ________
Next in sequence.

DESCRIPTION OF WHAT IS WRONG:
Nothing

PROBABLE CAUSE:
End of job routine is printing out the differences between the input totals and the accumulated totals.

RECOVERY PROCEDURES:
Nonzero figures must be accounted for and corrected.

COMMENTS:

SCORESHEET
DATE
INITIALS

36
IBM 1130 MACHINE SETUP SHEET

<table>
<thead>
<tr>
<th>PROGRAM NAME:</th>
<th>Calculations &amp; Payroll Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM NUMBER:</td>
<td>PAY04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROGRAM DESCRIPTION:</th>
<th>APPROXIMATE RUNNING TIME:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PRINTER</th>
<th>TYPE OF PAPER</th>
<th>NO. OF COPIES</th>
<th>CARRIAGE TAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard</td>
<td>1</td>
<td>Standard</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISKS</th>
<th>DRIVE NUMBER:</th>
<th>CARTRIDGE ID:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>Payroll</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWITCH SETTINGS</th>
<th>SWITCH</th>
<th>SWITCH</th>
<th>SWITCH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>14</td>
<td>15</td>
<td>None</td>
</tr>
<tr>
<td>UP</td>
<td></td>
<td>UP</td>
<td>UP</td>
</tr>
<tr>
<td>DOWN</td>
<td></td>
<td>DOWN</td>
<td>DOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INPUT CARDS</th>
<th>Switch 14 to change maximum check amount (and turn off) switch 15 to change check number and week number (and turn off)</th>
</tr>
</thead>
</table>

For one plant

WEEEKLY EMPLOYEE CARDS

CONTROL TOTALS

\[ XEQ PAY04 \]

\[ JOB \]

SOURCE OF INPUT:
1. Card input from a successful PAY16 edit run.
2. Disk must be payroll disk from files.

DISPOSITION OF OUTPUT:
1. Control totals to file D
2. Details to file D
3. Disk to storage
4. Payroll register to payroll section

FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS
PAY05: CHECK WRITING

Accounting Controls

Disk-stored totals -- gross ($) and net ($) -- are balanced to machine-calculated total of checks for zero-balance test. This should also be compared with the adding machine tape of checks.
IBM 1130 ERROR RECOVERY SHEET

JOB: Payroll System
PROGRAM NAME: PAY05
PROGRAMMER NAME: C. R. Kluck

MESSAGE TYPED:
CHECK CCI AND C080 ON FIRST CARD

PAUSE - DISPLAYED IN ACCUM:
0 0 0 1

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT 99999

DESCRIPTION OF WHAT IS WRONG:
1. Card column 1 has an invalid plant number
or
2. Card column 80 is not zero

PROBABLE CAUSE:
Either a blank card or a data card has been placed in front of the deck

RECOVERY PROCEDURES:
Clear the card reader, place a proper header card at the front of the deck. Ready the card reader and press Program Start on the console.

COMMENTS:

SCORESHEET

DATE
INITIALS
### IBM 1130 ERROR RECOVERY SHEET

#### JOB
Payroll System

#### PROGRAM NAME
PAY05

#### PROGRAMMER NAME
C.R. Elick

<table>
<thead>
<tr>
<th>COMPANY NAME</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHECK NO. XXXXX</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEEK NO. X</td>
</tr>
<tr>
<td>WEI-XXXXX</td>
</tr>
<tr>
<td>CHECK MAX XXXXX</td>
</tr>
</tbody>
</table>

#### MESSAGE TYPED:
4-4(44.146r441e.'l'el'seek "A/0.

#### PAUSE - DISPLAYED IN ACCUM:
[ ] [ ] [ ] [ ]

#### AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT
Next in sequence

#### DESCRIPTION OF WHAT IS WRONG:
Operator option to change constants.

#### PROBABLE CAUSE:
Program allows for this possibility

#### RECOVERY PROCEDURES:
Follow machine typed instructions.

#### COMMENTS:

#### SCORESHEET

<table>
<thead>
<tr>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
IBM 1130 ERROR RECOVERY SHEET

<table>
<thead>
<tr>
<th>JOB</th>
<th>PROGRAM NAME</th>
<th>PROGRAMMER NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payroll System</td>
<td>PAY05</td>
<td>C.R. Click</td>
</tr>
</tbody>
</table>

**MESSAGE TYPED:**

<table>
<thead>
<tr>
<th>ENTER CLOCK NO.</th>
<th>PAUSE - DISPLAYED IN ACCUM:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NONE</td>
</tr>
</tbody>
</table>

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT:** 500

**DESCRIPTION OF WHAT IS WRONG:**

Waiting for keyboard input.

**PROBABLE CAUSE:**

Operator intervention to reprint check(s).

**RECOVERY PROCEDURES:**

Enter the four-digit clock number for the check(s) to be reprinted. When all of the numbers have been entered, and this condition occurs, turn off switch zero and press EOF.

**COMMENTS:**

This routine allows for reprinting of torn or misaligned checks.

**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INITIALS</th>
</tr>
</thead>
</table>
# IBM 1130 ERROR RECOVERY SHEET

**JOB:** Payroll System  
**PROGRAM NAME:** PAY05  
**PROGRAMMER NAME:** C.R. Elick

<table>
<thead>
<tr>
<th>MESSAGE TYPED:</th>
<th>PAUSE - DISPLAYED IN ACCUM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;DEDUCTION MAY NOT MADE FOR XXXX.&quot;</td>
<td>NONE</td>
</tr>
</tbody>
</table>

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT:** 550

**DESCRIPTION OF WHAT IS WRONG:**
The gross amount of the check for this employee was not sufficient to allow this authorized deduction.

**PROBABLE CAUSE:** Employee did not work a full week.

**RECOVERY PROCEDURES:**
Notify your supervisor.

**COMMENTS:** This is to be reported to the employee.
# IBM 1130 ERROR RECOVERY SHEET

## Job
**Payroll System**

## Program Name
**PAY05**

## Programmer Name
**C.R. Kick**

### Message Typed:
**NONE**

### Pause - Displayed in Accum:
- 0002

### After Pause, Control Transfers to Statement:
91

### Description of What is Wrong:
Pause to allow the alignment of checks before printing the second time.

### Probable Cause:
Switch 15 has been set by the operator.

### Recovery Procedures:
Press Program Start to continue. When checks are aligned, turn off Switch 15.

### Comments:
Switch 15 will halt program for any emergency (form, ribbon, etc.).

### Scoresheet

<table>
<thead>
<tr>
<th>Date</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Initials</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IBM 1130 ERROR RECOVERY SHEET

<table>
<thead>
<tr>
<th>JOB</th>
<th>Payroll System</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM NAME</td>
<td>PAY05</td>
</tr>
<tr>
<td>PROGRAMMER NAME</td>
<td>C.R.Elick</td>
</tr>
</tbody>
</table>

MESSAGE TYPED:

**NONE**

PAUSE - DISPLAYED IN ACCUM:

0 0 0 3

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT 93

DESCRIPTION OF WHAT IS WRONG:

> Pause to allow the alignment of checks before printing the third line.

PROBABLE CAUSE:

Switch 15 has been set by the operator

RECOVERY PROCEDURES:

*Press Program Start to continue. When checks are aligned, turn off switch 15.*

COMMENTS:

SCORESHEET

<table>
<thead>
<tr>
<th>DATE</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

44
**IBM 1130 ERROR RECOVERY SHEET**

<table>
<thead>
<tr>
<th>JOB</th>
<th>PROGRAM NAME</th>
<th>PROGRAMMER NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payroll System</td>
<td>PAYOS</td>
<td>C.R. Klick</td>
</tr>
</tbody>
</table>

**MESSAGE TYPED:**

NONE

**PAUSE - DISPLAYED IN ACCUM:**

0 0 0 4

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT:**

95

**DESCRIPTION OF WHAT IS WRONG:**

Pause to allow the alignment of checks before printing the fourth line.

**PROBABLE CAUSE:**

Switch 15 has been set by the operator.

**RECOVERY PROCEDURES:**

Press Program Start to continue. When checks are aligned, turn off switch 15.

**COMMENTS:**


**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# IBM 1130 ERROR RECOVERY SHEET

## JOB
- **Payroll System**

## PROGRAM NAME
- **PAY05**

## PROGRAMMER NAME
- **C.R. Click**

### MESSAGE TYPED:
- **NONE**

### PAUSE - DISPLAYED IN ACCUM:
- **0 0 0 5**

### AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT
- **700**

### DESCRIPTION OF WHAT IS WRONG:
- Pause to allow the alignment of checks before printing the first line.

### PROBABLE CAUSE:
- Switch 15 has been set by the operator.

### RECOVERY PROCEDURES:
- Press Program Start to continue. When checks are aligned, turn off switch 15.

### COMMENTS:
- 

---

<table>
<thead>
<tr>
<th>SCORESHEET</th>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IBM 1130 ERROR RECOVERY SHEET

JOB Payroll System
PROGRAM NAME PAY05
PROGRAMMER NAME C.R.Evick

MESSAGE TYPED:
REGISTER CHECK NO XXXXX
DOES NOT AGREE WITH
THIS RUN CHECK
NO XXXXX

PAUSE - DISPLAYED IN ACCUM:
NONE

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT 802

DESCRIPTION OF WHAT IS WRONG:
The final check number from the payroll
register stored on disk does not agree with
the final check number from this run.

PROBABLE CAUSE:
Additional check numbers were used

RECOVERY PROCEDURES:
The difference must be accounted for
in rewritten checks. If too few checks were
written, attempt to write the missing checks
by reexecuting the program using the
reprinting option, switch zero. If this fails,
notify your supervisor.

COMMENTS:

SCORESHEET
DATE
INITIALS

47
IBM 1130 ERROR RECOVERY SHEET

JOB Payroll System
PROGRAM NAME PAY05
PROGRAMMER NAME C.R. Click

MESSAGE TYPED: CHECK NUMBERS AGREE

PAUSE - DISPLAYED IN ACCUM:

N O N E

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT 802

DESCRIPTION OF WHAT IS WRONG:

Nothing

PROBABLE CAUSE:

End-of-job routine

RECOVERY PROCEDURES:

None

COMMENTS:

SCORESHEET

DATE

INITIALS
**IBM 1130 ERROR RECOVERY SHEET**

**JOB** Payroll System  
**PROGRAM NAME** PAY05  
**PROGRAMMER NAME** C.R. Elick

<table>
<thead>
<tr>
<th>MESSAGE TYPED:</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGISTER TOTALS XXXXXXXX X XXXXXXX</td>
</tr>
<tr>
<td>CHECK TOTALS XXXXXXXX X XXXXXXXX</td>
</tr>
<tr>
<td>DIFFERENCES XXXXXXXX X XXXXXXXX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAUSE – DISPLAYED IN ACCUM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
</tr>
</tbody>
</table>

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT** Next

Next in sequence.

**DESCRIPTION OF WHAT IS WRONG:**

1. Disk-stored register totals.
2. Totals accumulated during the run.
3. Differences between the two

**PROBABLE CAUSE:**

End of job.

**RECOVERY PROCEDURES:**

Nonzero differences must be accounted for and corrected.

**COMMENTS:**


**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IBM 1130 MACHINE SETUP SHEET

PROGRAM NAME: Check Writing  PROGRAM NUMBER: PAY05

PROGRAM DESCRIPTION: APPROXIMATE RUNNING TIME:

<table>
<thead>
<tr>
<th>PRINTER</th>
<th>TYPE OF PAPER</th>
<th>NO. OF COPIES</th>
<th>CARRIAGE TAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Checks</td>
<td></td>
<td>Checks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISKS</th>
<th>DRIVE NUMBER:</th>
<th>CARTRIDGE ID:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>Payroll</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWITCH SETTINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWITCH</td>
</tr>
<tr>
<td>UP</td>
</tr>
<tr>
<td>DOWN</td>
</tr>
</tbody>
</table>

Switch 0 is used to make checks reprint when they are not correct.
Switch 14 is used to set the maximum check amount.
Switch 15 is used to set the check number to start with, and to stop the system to align the printer.

SOURCE OF INPUT:
1. Control totals from file 0
2. Disk must be payroll disk from files

DISPOSITION OF OUTPUT:
1. Paychecks to employees
2. Disk for control totals to be used with PAY06

FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS
PAY06: CHECK REGISTER

Accounting Controls

Plant total (net) from payroll register is balanced to total on check register, and check register total (net $) is balanced to adding machine tape of checks.
IBM 1130 Error Recovery Sheet

**JOB:** Payroll System
**PROGRAM NAME:** PAY06
**PROGRAMMER NAME:** C.R. Klick

**MESSAGE TYPED:**
CHECK CC1 AND CCBO ON FIRST CARD

**PAUSE - DISPLAYED IN ACCUM:** 00001

**AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT:** 99999

**DESCRIPTION OF WHAT IS WRONG:**
1. Card column 1 has an invalid plant number.
or
2. Card column 80 is not zero.

**PROBABLE CAUSE:**
Either a blank card or a data card has been placed in front of deck.

**RECOVERY PROCEDURES:**
Clear the card reader. Place a proper header card in the front of the deck. Ready the reader and press Program Start on the console.

**COMMENTS:**

**SCORESHEET**

<table>
<thead>
<tr>
<th>DATE</th>
<th>INITIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IBM 1130 MACHINE SETUP SHEET

PROGRAM NAME: Check Register
PROGRAM NUMBER: PAY06

PROGRAM DESCRIPTION:

APPROXIMATE RUNNING TIME:

<table>
<thead>
<tr>
<th>PRINTER</th>
<th>TYPE OF PAPER</th>
<th>NO. OF COPIES</th>
<th>CARRIAGE TAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard</td>
<td>1</td>
<td>Standard</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISKS</th>
<th>DRIVE NUMBER:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CARTRIDGE ID:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Payroll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWITCH SETTINGS</th>
<th>SWITCH</th>
<th>SWITCH</th>
<th>SWITCH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UP</td>
<td>UP</td>
<td>UP</td>
</tr>
<tr>
<td></td>
<td>DOWN</td>
<td>DOWN</td>
<td>DOWN</td>
</tr>
</tbody>
</table>

INPUT CARDS

SOURCE OF INPUT:
1. Disk & control totals from PAY05

DISPOSITION OF OUTPUT:
1. Check register to payroll section
2. Control totals to file D
3. Disk is returned to storage

FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS
PAY09: 941 REPORT

Accounting Controls

941 total per plant (Gross $) is balanced to general ledger.
IBM 1130 ERROR RECOVERY SHEET

JOB: Payroll System
PROGRAM NAME: PAY22
PROGRAMMER NAME: C.R. Klick

MESSAGE TYPED:

PAUSE - DISPLAYED IN ACCUM:

NONE

AFTER PAUSE, CONTROL TRANSFERS TO STATEMENT: None

DESCRIPTION OF WHAT IS WRONG:

None

PROBABLE CAUSE:

None

RECOVERY PROCEDURES:

None

COMMENTS:

None

SCORESHEET

DATE
INITIALS
<table>
<thead>
<tr>
<th>IBM 1130 MACHINE SETUP SHEET</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROGRAM NAME:</strong> 941 REPORT</td>
</tr>
<tr>
<td><strong>PROGRAM DESCRIPTION:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TYPE OF PAPER</th>
<th>NO. OF COPIES</th>
<th>CARRIAGE TAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>941 FORMS</td>
<td></td>
<td>941 TAPE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRINTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>941 FORMS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRIVE NUMBER:</td>
</tr>
<tr>
<td>CARTRIDGE ID:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWITCH SETTINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWITCH</td>
</tr>
<tr>
<td>NONE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INPUT CARDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>For one plant</td>
</tr>
<tr>
<td>MORE PLANTS</td>
</tr>
<tr>
<td>PLANT ACCOUNT NO. CARD</td>
</tr>
<tr>
<td>PLANT CITY-STATE CARD</td>
</tr>
<tr>
<td>PLANT ADDRESS CARD</td>
</tr>
<tr>
<td>PLANT NAME CARD</td>
</tr>
<tr>
<td>PLANT HEADER CARD</td>
</tr>
<tr>
<td>11 XEQ PAY09</td>
</tr>
<tr>
<td>11 JOB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOURCE OF INPUT:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- Plant information cards from file E</td>
</tr>
<tr>
<td>2- Payroll disk from storage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISPOSITION OF OUTPUT:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- 941 report to government</td>
</tr>
<tr>
<td>2- Disk is returned to storage</td>
</tr>
<tr>
<td>3- Plant information cards to file E</td>
</tr>
</tbody>
</table>

FOR PAUSES AND ERROR MESSAGES SEE ERROR RECOVERY SHEETS
Section 40: CONVERSION

CONTENTS

<table>
<thead>
<tr>
<th>Introduction</th>
<th>40.01.00</th>
<th>Preparing for Conversion</th>
<th>40.20.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning for Conversion</td>
<td>40.10.00</td>
<td>Conversion Methods</td>
<td>40.30.00</td>
</tr>
</tbody>
</table>
INTRODUCTION

For each application, there will come a day when all your programs are written and tested, and you will be ready to convert from your old system to the new. Will you be ready? Not unless you have planned and prepared for conversion. Conversion involves three major elements, of which the conversion itself is the last step.
PLANNING FOR CONVERSION

As has been stressed, the first step is planning. This involves two basic items:

1. Make a schedule indicating when you will start conversion of each application and when conversion will be complete. After you have completed conversion of your first application, you will have a better feel for what is involved, and will want to review the schedule for the remaining areas. You may also want to reevaluate the conversion techniques you chose originally.

2. Decide which conversion technique you will use for each application area. As above, you will want to periodically reexamine your decisions as you become more experienced with each technique.
PREPARING FOR CONVERSION

After making up conversion schedules and choosing techniques, you should be able to see what must be done to prepare for the actual conversion. Ask yourself these questions:

1. Is the old system documented accurately and completely? (See Section 10.) If it isn't, a smooth conversion will be difficult.

2. Can the controls of the two systems be compared? If not, it will be difficult to compare the two systems. The new system should have the same controls as the old, and you may even want to add controls to the old system to ease conversion.

Such controls as grand totals, subtotals, document counts, etc., will bring quick attention to discrepancies between the two systems.

3. Is everyone who is involved in the conversion familiar with both the old and new systems? Misunderstandings regarding the differences between the old and new can seriously interfere with and delay the successful completion of even the best planned conversion effort. Communications should be maintained with the people involved during the entire application design and program development phases. A few weeks before the conversion period, all those who will be involved indirectly or directly in preparing input or using output from the new system should be taught both systems, in general—and their particular areas of responsibility, in detail.
CONVERSION METHODS

There are three common methods for conversion:

1. Parallel operation. With this method, the same transactions are entered into both the old and the new systems, and the controls are compared. This process is continued over a predetermined (usually short) period of time, until a responsible executive is satisfied that the new results are accurate.

Make sure that the time period of parallel operation is one during which a wide variety of transactions occur. Large volume is not important, but variety is, since you want to test as many aspects of the new systems as possible. Pick a slow time in your business cycle to effect conversion.

Before starting parallel operations, obtain a clear understanding of what is to be checked, and by whom. Since additional personnel or man-hours will be needed during this period, avoid conflicts with vacation and holiday schedules.

As far ahead of the parallel period as possible, the personnel who will be preparing the input cards for the new system should gain experience in using the new input document and card formats. This is one of the most common areas of difficulty, and many "computer" mistakes are eventually traced back to faulty document preparation, accumulation of controls, or card punching. Often it is possible to use new formats exclusively some time before the computer system arrives, by preparing cards in the computer-required formats and then reproducing them into the old formats for use by the current system.

Parallel operations often encounter problems that result from significant differences between the procedures used in the old system and those in the new. It may be quite difficult to compare results produced by the two systems, since the important totals in the new system may not have been prepared previously. Or you may find it possible to print reports in a desirable sequence which is not feasible currently, but which will make it impractical to cross-check line-items against reports in the old sequence.

Another problem inherent in parallel operations is the doubled probability of errors. There are twice as many chances for errors to occur, and when making up a schedule, you must consider the time spent in tracking errors down and deciding which system, if either, was right.

2. Pilot Operation. In pilot as in parallel operation, an application is run under both the old and the new systems. The difference lies in selecting only one or a few easily observed locations or departments within the company, and performing the operation only for those sections rather than for the entire company. The same care must be taken in setting up controls, scheduling the period during which the pilot operation is to take place, and training those who prepare the input. In regard to this last problem, the pilot method offers a training ground for those who prepare and punch the data, by allowing different people to get experience every day or every few hours.

Care should be taken in determining which part of the job is selected for pilot running. It should be completely independent and self-contained, if possible. Therefore, pilot operations may be the ideal choice for organizations that are divided into fairly independent units or locations. In any case, the effect of the pilot run on departments other than the data processing department must be carefully analyzed, and those who are affected should be notified well ahead of time.

Again, you must carefully establish who is to do what and when, if an adequate analysis of the progress and success of the operation is to be made.

3. One-time cutover. As of a given date, the old system is discontinued and the new system is put into operation. Careful planning is necessary to make the transition smooth. For one thing, files can be built up during a fairly extensive period beforehand and checked with control figures for accuracy and completeness while being created. A master file of customers can be card-punched during the month before the preparation of statements. Alternatively, only new customers' cards can be punched, while operations are performed on the old file to convert them into the new format. Then both the old and new cards are merged at month end to create an updated master file ready for use by the new system. It is often desirable to write one-time programs to do these file conversions.

Where some data is to be recoded, or coded for the first time, as in the assignment of a new or better set of customer numbers, you should get the job done and checked out in advance.

Another way of smoothing the cutover is to maintain control procedures that will be required for both the old and new systems some time before the critical date. This will eliminate the possibility of errors in the execution of these procedures.
Cutovers are never truly "one-time" in the sense that no parallel or pilot operations are performed. The difference is in the time at which these operations are done. With the cutover method, parallel and pilot operation take place with data that has already been processed. For instance, after an accounts receivable procedure has been processed under a current system, the entire procedure is run again on the computer. Controls are checked and errors are cleared up. The accounts receivable may then be run once more on the computer, and this process may be repeated, perhaps over more than one month, until management is satisfied that it is running correctly.

Although some double manpower requirements may be eliminated by using the one-time cutover method, extra man-hours will still be needed—for example, when a weekend immediately precedes the cutover date, or when card files are being converted from one format to another.

***

You can see that no one of the conversion methods discussed here stands alone and independent of the others. Use the elements of each that suit your situation, but develop a realistic plan that will consider these factors:

1. Manpower must be available at the right time to manipulate old data into new formats.
2. Control procedures must be developed and, if possible, tested ahead of time.
3. Detailed document preparation and card-punching procedures must be developed, and a reasonable amount of time must be reserved to practice them before conversion.
4. Procedures must be written for the one-time aspects of the job, and manpower must be available at the right time to do so.
5. The word must be spread; education for those in other departments must be done thoughtfully and carefully.

It is almost impossible to plan a conversion too carefully.
Section 45: 1130 COMPUTING SYSTEM

CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Subsections</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1130</td>
<td>45</td>
<td>01</td>
</tr>
</tbody>
</table>

Introduction ........................................ 45.01.00
The 1131 CPU ....................................... 45.05.00
  Console Printer and Keyboard ................. 45.05.10
  Data Switches .................................. 45.05.20
  Console Display Lamps ....................... 45.05.30
Disk Storage ....................................... 45.10.00
Printers ............................................ 45.15.00
Card Readers and Punches ...................... 45.20.00
Paper Tape Readers and Punches ............ 45.25.00
Plotter ............................................. 45.30.00
Graphic Display ................................ 45.35.00
Optical Readers .................................. 45.40.00
Storage Access Channel ...................... 45.45.00
Teleprocessing ................................. 45.50.00
The 1130 Configurator ......................... 45.55.00
INTRODUCTION

The IBM 1130 Computing System is a flexible, modular, and modern data processing system. In capability, it can range from a small paper-tape-oriented system to a large, multiple-disk system, with a powerful complement of input/output devices. This section describes the system components in general terms, stressing their potential use, the various possible combinations of units, and their corresponding throughput capabilities. For more detail see IBM 1130 Functional Characteristics (A26-5881) and IBM 1130 Input/Output Units (A26-5890).
1131 CENTRAL PROCESSING UNIT

The 1131 CPU is available with three options:

- With or without disk storage
- 3.6- or 2.2-microsecond core storage access time
- 4096, 8192, 16,384, or 32,768 words (16 bits) of core storage

Although this yields 16 possible combinations, only 9 are currently available, as shown in Figure 45.1.

All 1131 CPUs, regardless of model, have as standard components:

- A console printer
- A console keyboard
- 16 data switches
- Console display lamps
- Processing functions (index registers, indirect addressing, multiply/divide, etc.)

<table>
<thead>
<tr>
<th>Without Disk Storage</th>
<th>With Disk Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6 Microsecond</td>
<td>3.6 Microsecond</td>
</tr>
<tr>
<td>2.2 Microsecond</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core Storage Capacity</th>
<th>4K</th>
<th>8K</th>
<th>16K</th>
<th>32K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Designation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1A 1B 2A 2B 2C 2D 3B 3C 3D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 45.1. Available 1131 Processing Unit Configurations

The first four components are described below in more detail, since they may be directly used by the programmer.
Console Printer and Keyboard

The console printer is a modified SELECTRIC® typewriter printer and can provide output at 15.5 characters per second. If it is the primary (only) printing device on the 1130, it must be used for all printed output; however, if the system includes an 1132 or 1403 Printer, the console printer will normally be used only for error messages, operator instructions, etc.

The console keyboard resembles a standard typewriter keyboard and allows the 1130 operator to enter data into the system.

Because it is a manually oriented device, the use of the keyboard will usually be limited to small quantities of data (today's date, starting check number, etc.), with the card or paper tape readers used for more voluminous data.
Data Switches

Mounted on the front face of the console printer is a row of 16 toggle switches, called data switches. They may be used by the programmer for the entry of yes-or-no type information into the system. For example, one payroll program might handle both factory workers and office workers, with each group processed separately. The program could be written to read, say, data switch 6, treating the input time cards as factory workers if that switch is on, and as office workers if it is off.

Other uses of the console switches are to bypass certain portions of a program, activate the FORTRAN TRACE, etc.
Console Display Lamps

Above the console printer is a panel containing a large number of indicator lamps (or lights). These lights indicate the internal status of the 1130 Computing System. While most are of little use to the average programmer, he does have access to one set of lamps: the accumulator.

The accumulator is displayed as a series of 16 numbers, in four groups of four, which are either illuminated (backlighted) or not. For example, suppose the accumulator indicates the status shown below, where the underlined numerals are lit:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Since the accumulator displays a binary number, this example means that it contains 0100 1000 1000 1001, or 18569 in decimal. An easier way to represent the number is to use the hexadecimal notation, where each group of four "bits" is taken as a hexadecimal number, obtaining 4889. (For further detail on number systems, see Appendix A of A26-5881.)

The programmer can use the accumulator display feature by appending a four-digit number (from 0001 to 9999) to the FORTRAN PAUSE or STOP statements. If the programmer inserts a PAUSE 3322 statement in his program, the CPU will pause and display 3322 in the accumulator (as a hexadecimal number) when it executes the PAUSE statement:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

If the program contains many PAUSEs, each may be given a different number, and the operator can determine which PAUSE caused the CPU to halt its operations.

This facility is useful for indicating error conditions, tracing progress through a program, etc.
DISK STORAGE

Models 2 and 3 of the 1131 CPU contain a disk storage drive as an integral part of the console unit. In addition, these models may contain up to four additional disk drives, mounted in separate enclosures (the IBM 2310 Disk Storage).

Each disk drive will hold one IBM 2315 Disk Cartridge. Because the cartridges are removable, the user may have an unlimited amount of data on them; only one, however, may be mounted in a disk drive at any one time.

The 2315 cartridge consists of a single metal plate, coated on both sides with magnetic material and enclosed in a plastic container. When mounted in an activated disk drive, the metal plate is driven through a clutch mechanism at 1500 revolutions per minute. The recording plate never leaves its container, as it does in the case of some other disk devices.

Each cartridge is divided into 200 cylinders, in concentric circles, with each cylinder further divided into eight sectors — four on the top surface and four on the bottom. Since each of the 1600 sectors contains 320 words, each disk cartridge can hold 512,000 words.

Data is read or written on the disk by two read-write heads attached to a movable arm. One setting of the arm gives the 1130 access to one cylinder, or eight sectors. One head reads (or writes) the top four sectors; the other, the bottom four sectors. The two heads cannot move independently, since they are fixed to the same arm.

Because one setting of the arm gives access to only one cylinder, the arm must be moved in order to read or write on a different cylinder. For example, to read from cylinder 10 and then write on cylinder 15, the arm must move, or "seek", from cylinder 10 to cylinder 15. Since the arm moves in steps of one or two cylinders, this would require two 2-cylinder moves (from 10 to 12, and from 12 to 14) and one 1-cylinder move (from 14 to 15).

Each move, whether one or two cylinders in length, takes 15 milliseconds (0.015 seconds). A five-cylinder "seek", as shown above, would take 45 milliseconds (15+15+15). A six-cylinder seek would take the same length of time.

Because this can be a relatively lengthy operation (compared with other 1130 functions), you should attempt to minimize the need for disk arm movement. Many hints on how to do this are given later in the manual (Sections 55, 60, 65, 70, 80, 85, and 90).

Having reached the desired cylinder, the arm takes another 25 milliseconds to stabilize. After the stabilization period, data may be read or written; because the disk is rotating, however, it will be quite unusual for the desired sector to be passing under the read/write head at the precise time you want it. You will have to wait an average of half a revolution (20 milliseconds) for the sector to reach the heads, and then 10 more milliseconds for it to actually be read or written.

Figure 45.2 gives some examples of how long it takes to move n cylinders, then read one sector.

<table>
<thead>
<tr>
<th>Move This Many Cylinders</th>
<th>Seek Time</th>
<th>Stabilization Time</th>
<th>Average Rotational Delay Time</th>
<th>Read or Write</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>1 or 2</td>
<td>15</td>
<td>25</td>
<td>20</td>
<td>10</td>
<td>70</td>
</tr>
<tr>
<td>3 or 4</td>
<td>30</td>
<td>25</td>
<td>20</td>
<td>10</td>
<td>85</td>
</tr>
<tr>
<td>5 or 6</td>
<td>45</td>
<td>25</td>
<td>20</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>199 or 200</td>
<td>1500</td>
<td>25</td>
<td>20</td>
<td>10</td>
<td>1995</td>
</tr>
</tbody>
</table>

Figure 45.2.
PRINTERS

In addition to the console printer, which is standard, the 1130 system can be configured with four combinations of line printers:

- No line printer
- An IBM 1132 Printer
- An IBM 1403 Printer
- Both an 1132 and 1403 Printer

The 1132 and the 1403 Printers have many mechanical differences, but the primary difference is in printing speed. Both print a line at a time, 120 characters wide; both have a carriage control tape that controls the vertical movement of forms.

The 1132 has a rated speed of 110 lines per minute when printing purely numeric and 80 lines per minute when printing alphanumerical information.

The 1403 prints both numeric and alphanumerical information at the same speed; 340 lines per minute (maximum) in the case of the 1403 Model 6, 600 lines per minute (maximum) for the Model 7.
IBM 1403 Printer
CARD READERS AND PUNCHES

Five card readers and/or punches are available for attachment to the 1130 system.

The IBM 1142 Card Read Punch, Model 6, reads and punches cards, with all input from a single hopper. It reads at a rated speed of 300 cards per minute, and punches at 80 card columns per second.

The IBM 1442 Card Read Punch, Model 7, is similar to the Model 6, but faster, reading at 400 cards per minute and punching at 160 columns per second.

The IBM 1442 Card Punch, Model 5, cannot read cards; it can only punch. Its punching speed is 160 columns per second.

The IBM 2501 Card Reader, Model A1, will read cards at a rated maximum speed of 600 per minute. It is not able to punch cards.

The IBM 2501 Card Reader, Model A2, is similar to the A1, but operates at 1000 cards per minute (maximum).

Disregarding speeds for the moment, there are four combinations of card readers and/or punches for the 1130:

1. No card readers or card punches
2. An IBM 1442 Card Read Punch
3. An IBM 2501 Card Reader and the IBM 1442 Card Punch, Model 5
4. An IBM 2501 Card Reader and an IBM 1442 Card Read Punch

Aside from speed, the main difference between combinations is capability — the number of card paths available.

Configuration 2 (1442 Model 6 or 7) gives the user only one card path. This means that cards to be read and cards to be punched must both be placed in the same input hopper in the proper order.

Configuration 3 (2501 and 1442 Model 5) has separate paths for reading and punching, which simplifies programming and operating in certain types of applications.

Configuration 4 (2501 and 1442 Model 6 or 7) also has two card paths, differing from configuration 3 in that one path can both read and punch. In certain applications this can be very useful. For example, you could put a master card deck in one reader and a detail deck in the other reader, eliminating the need to collate (merge) the two together.
PAPER TAPE READERS AND PUNCHES

The 1130 system may include the IBM 1134 Paper Tape Reader and/or the IBM 1055 Paper Tape Punch.

The 1134 reads punched tape at 60 characters per second; the 1055 punches tape at 15 characters per second.
PLOTTER

For hard-copy graphic output, the IBM 1627 Plotter may be attached to the 1130 system. Bar charts, flowcharts, organization charts, engineering drawings, and maps, in addition to graphs or drawings that depict financial, scientific, or technical data, can be plotted on the 1627 Plotter.

Two models are available:

Model 1
Plotting area: 11 inches by 120 feet
Step size: 1/100-inch increments
Speed: 300 steps per second

Model 2
Plotting area: 29-1/2 inches by 120 feet
Step size: 1/100-inch increments
Speed: 200 steps per second

The 1627 can plot curves, straight lines, alphanumeric headings, etc., by a series of steps in which either the pen, the drum, or both, move in 1/100-inch increments.
GRAPHIC DISPLAY

A second means of graphic display may be obtained by attachment of the IBM 2250 to the 1130 system. The 2250 is an electronic (cathode ray tube) device, and therefore capable of faster speeds than the 1627 Plotter, a mechanical device. A "light pen" enables the operator to communicate with the system by interacting with the display on the face of the tube.
OPTICAL READERS

The IBM 1231 Optical Mark Page Reader reads positional marks made by an ordinary lead pencil on paper documents, such as test scoring sheets, etc. The data contained on these documents can be read into the 1130 system at a rate of 2000 sheets per hour.

The 1231 is especially suited for applications such as examination grading, surveys, order entry, etc., where variable information may be entered by hand on preprinted forms.
STORAGE ACCESS CHANNEL

The storage access channel provides an input/output "path" that allows nonstandard components to be added to the 1130 system. These components may be IBM-supplied, or user-supplied. Since the SAC is merely a general purpose input/output channel, control of the nonstandard component must be handled by user-supplied hardware and/or programming.
TELEPROCESSING

By means of the Synchronous Communications Adapter (SCA), the 1130 may communicate, over telephone lines, with another 1130, an IBM System/360, and/or other devices.
THE 1130 CONFIGURATOR

The accompanying schematic is a copy of the 1130 Configurator (A26-5915).
Section 50: 1130 DISK MONITOR SYSTEM

CONTENTS

GENERAL ...................... 50.01.00
GENERAL

This section consists of a general discussion of the 1130 Disk Monitor System and serves to introduce the next three sections:

- Job Management -- how the Monitor helps you achieve smooth, orderly, automatic transition from each job to the next.
- Disk Management -- how the Monitor helps you manage the disk and use it efficiently.
- Core Storage Management -- how the Monitor allows you to make the most effective use of the available core storage.

If your 1130 does not have disk capability, you cannot use the Monitor, and you may skip over this and the succeeding three sections.

The 1130 Disk Monitor System is a disk-oriented operating system that allows the user to assemble, compile, and/or execute individual programs or groups of programs with a minimum of operator intervention. Jobs to be performed are stacked and separated by control records that identify the operation to be performed.

The Monitor System consists of five distinct but interdependent programs (see Figure 50.1):

- Supervisor Program
- Disk Utility Program
- Assembler Program
- FORTRAN Compiler
- Subroutine Library

The supervisor program provides the necessary control for the stacked-job concept. It reads and analyzes the monitor control records, and transfers control to the proper program.

The Disk Utility Program is a group of routines designed to assist the user in storing information (data and programs) on the disk, and in retrieving and using the information stored.

The Assembler program converts user-written symbolic-language source programs into machine-language object programs.

The FORTRAN compiler converts user-written FORTRAN-language source programs into machine-language object programs.

The Subroutine Library contains subroutines for data input/output, data conversion, and arithmetic functions.

The Monitor System coordinates program operations by establishing a communications area in core storage that is used by the various programs making up the Monitor System. It also guides the transfer of control between the various monitor programs and the user's programs. Operation is continuous and setup time is minimized, thereby effecting substantial time saving and allowing greater programming flexibility. The complete Monitor System resides on disk storage, but only those routines or programs required at any one time are transferred to core storage for execution. This feature minimizes the core storage requirements and permits segmenting of long programs.

In addition to providing you with an efficient job-to-job transition system, the 1130 Disk Monitor System significantly reduces the amount of programming you must do. This is made possible through the sharing of common subroutines by unrelated programs. For example, input/output or conversion operations are required by most user programs, whether the programs are written in the Assembler Language or in FORTRAN. IBM provides a library of subroutines to handle such operations as an integral part of the Monitor System.

The Disk Utility Program (DUP) facilitates development of a library of user programs. Programs can be stored on cards or paper tape, as is customary in installations without disk storage. With disk storage, programs can also be stored directly on the disk. The disk-stored programs and data are referred to by name when called for use. The Monitor System, through the use of a table known as the Location Equivalence Table (LET), can locate any user program, subroutine, or file by a table search for the name. Stored with the name is the amount of disk storage required by the program or data.

Any program that is added to the user's disk-stored programs is usually placed at the end of...
the other programs. If a program is deleted, the remaining program(s) are moved up on the disk in order to utilize disk storage effectively.

Section 55: THE MONITOR-JOB MANAGEMENT

CONTENTS

- Introduction ...................... 55.01.00
- Job and Subjob ..................... 55.10.00
- Stacked Jobs or the Input Stream .... 55.20.00
- Disk Cartridge ID Checking .......... 55.30.00
INTRODUCTION

The first function of the 1130 Disk Monitor System is Job Management -- helping you, the user, achieve a smooth, orderly transition from one job to the next. The Monitor is designed to accept a continuous stream of input, in the form of jobs and subjobs.
JOB AND SUBJOB

A job is defined as:
- A JOB card and all the following control records, source programs, object programs, and data, up to, but not including, the next JOB card.
- The processing that takes place from the detection of one JOB card (or paper tape record) until the detection of another JOB card.

A subjob is defined as:
- A monitor control record and all the following control records, source programs, object programs, and data, up to, but not including, the next monitor control record.
- The processing that takes place from the detection of one monitor control record (such as DUP card, FOR card, etc.) to the detection of another monitor control record.

A job is an independent unit of processing; a subjob is a unit of processing that is dependent on the subjob(s) preceding and/or following it. The successful completion of the job depends on the successful completion of each subjob within it. In some cases, a subjob is not attempted if the preceding subjobs have not been successfully completed.

The JOB control record defines the start of a new job. It causes the Supervisor to perform the job initialization procedure, which includes:

1. Initialization of constants, parameters, etc.
2. Setting of the temporary indicator if a T is present in column 8 of the control record. If set, all programs or data files stored in the User Area by DUP during the current job will be deleted automatically at the end of the job (that is, at the beginning of the next job).
3. The identification of the cartridge(s) to be used during the current job.
4. The definition of the cartridge on which the Core Image Buffer for the current job is to be found. Core image programs can be built faster if the CIB is assigned to a cartridge other than the systems cartridge. (This applies only to systems with two or more disk drives.)
5. The definition of the cartridge whose Working Storage is to be used by the Monitor system. (This applies only to systems with two or more disk drives.) Although all cartridges contain a Working Storage area, only one will be used by the Monitor (for its own purposes). Core image programs can be built faster if the system Working Storage is on some cartridge other than the systems cartridge. They can be built even faster if the CIB, the system Working Storage, and the monitor system itself are on separate cartridges. Assemblies are also faster if Working Storage is on a separate cartridge.
6. The starting of a new page. A skip to channel 1 is executed on the 1132 Printer or 1403 Printer; ten consecutive carriage returns are made on the console printer.
STACKED JOBS OR THE INPUT STREAM

Figure 55.1 shows a schematic view of a stack of three jobs:

JOB 1

- Translate an Assembler Language source program into an object program (subjob 1)
- Store the assembled object program (subjob 2)
- Execute the program (subjob 3)

JOB 2

- Store a program that had earlier been dumped onto cards (subjob 1)

JOB 3

- Compile a FORTRAN program (subjob 1)
- Execute it (subjob 2)

Here, the reason for the job/subjob concept can be seen clearly. If there were an error in subjob 1 of job 1, the assembly, you would not want to continue with the next two subjobs. The results would be meaningless.

If those first three items had been made jobs rather than subjobs, the Monitor would have tried to perform the second two tasks even though the first had failed. However, because they are all subjobs, an error condition encountered in any one subjob would cause the Monitor to abandon the remaining subjobs.
Figure 55.1. Stacked Job Input
DISK CARTRIDGE ID CHECKING

A second assist given you by the Monitor system is the checking of disk cartridge ID numbers. Every cartridge must have an ID number; if you so desire, you can request that the Monitor check each cartridge for a certain ID and alert you if the desired cartridges are not mounted.

For example, suppose you have placed a payroll data file on a particular cartridge, and have identified it as cartridge 6066. If you punch 6066 in columns 11 through 14 of the JOB card, the Monitor will read the cartridge ID from the disk on logical drive 0, and, if it is not 6066, you will be so informed with a message.

If you don’t care which cartridge is mounted (or, more likely, if you will check it yourself), those columns on the JOB card may be left blank.
Section 60: THE MONITOR-DISK MANAGEMENT

CONTENTS

Introduction ............................... 60.01.00
Disk Storage Layout ...................... 60.10.00
Introduction ............................... 60.10.01
Cylinder 0 ................................. 60.10.10
IBM Systems Area ......................... 60.10.20
Working Storage (WS) ............ 60.10.30
User Area (UA) ..................... 60.10.40
Fixed Area (FX) ................... 60.10.50
Summary .................................. 60.10.60
Increasing the Amount of Space
Available to the User .............. 60.20.00
Introduction ............................... 60.20.01
How Much Room Do I Have? ......... 60.20.10
How Can I Make More Space
Available? ............................... 60.20.20
Cylinder 0
IBM System's Area
Fixed Area
User Area/Working Storage
I/O Subroutines for Devices Not on
Your System
Computational Subroutines You Are
Unlikely to Use
Seldom-Used Programs and/or
Data
Unneeded User-Written Programs
and Data
Summary .................................. 60.20.30
The Disk Utility Program .............. 60.30.00
Introduction ............................... 60.30.01
Format of Material on the Disk ...... 60.30.02
Data Files
Programs and Subprograms
The Most Commonly Used DUP
Functions .................................. 60.30.20
Store a Program or Subprogram
in DSF Format
Store a Program in DCI (Core
Image) Format
Convert a DSF Program to DCI
Delete a Program or Subprogram
Dump a DSF Program or Subprogram
and Reload It
Dump a DCI (Core Image) Program
and Reload It
Dump a Data File and Reload It
Copy a Data File onto Another Area
on Same Disk
Defining and Modifying the Fixed
Area
Special Options — Multiple Disk
1130 Users ............................... 60.30.30
Copy a Data File onto Another Disk
Copy a Program onto Another Disk
Copy an Entire Disk onto Another
Disk
INTRODUCTION

Remember, effective management can make or break a good installation. This also applies to the disk portion of your 1130. Because the disk is such an integral part of your system, it is extremely important that you have the knowledge and ability to manage it effectively. This discussion of the disk, its layout, and how the Monitor helps you use it, will give you a good start toward effective disk management.

Effective use of your disk cartridges requires a certain amount of planning, especially if the number of applications on your 1130 is high, or is expected to grow. Some control must be exercised over what gets stored on a disk, and which disk cartridge is to be used for a particular job.

Each installation requires a certain minimum number of disk cartridges:
- At least one general purpose systems cartridge, with a complete Monitor system (FORTRAN and Assembler). It should only be used for testing, one-time applications, and other odd jobs.
- On multiple disk drive systems, at least one working or scratch disk for each disk drive over and above the first.
- One disk cartridge to be used for ordering and receiving programs from IBM. Some packages are not available in card form and can be obtained only by forwarding a cartridge to the Program Information Department. PID will place the package on your cartridge and return it to you.
- One disk cartridge (as required) for each of the major IBM applications programs to be used. For example, STRESS, COGO, LP-MOSS, and others each require all or most of a disk cartridge.
- One disk cartridge for each major application area, such as payroll, accounts payable, plant scheduling, highway design, etc. In some cases, two applications must share a disk because they both use the same data file, but such dual use should be avoided whenever possible.

Mixing of different applications on the same disk may lead to several complications, especially if different programmers are involved. For example:
1. Duplicate program and data file names may occur, with resulting confusion.
2. One program may inadvertently write into the disk data area of another program.
3. The amount of Working Storage is decreased more rapidly as each application area adds programs, subprograms, etc.

4. Run times may increase as data files are pushed further apart by the continuous storing and deleting of programs, data files, etc.
5. Overall control is diminished.

Before discussing disk storage management, several terms must be defined:

- Systems cartridge -- a cartridge that contains the 1130 Disk Monitor system. If your 1130 has only one disk drive, all your cartridges must be systems cartridges.
- Non-systems cartridge -- a cartridge that does not contain the monitor system. As implied above, such a cartridge would be of use only in installations with two or more disk drives.
- Master cartridge -- a systems cartridge that has been referenced by the cold start procedure, or by a Job card. The Monitor system on that cartridge will be the one in use until another cold start is initiated, or until a Job card is encountered that switches control to a different cartridge. Obviously, on a one-drive 1130 system, the one and only disk cartridge will be both a systems disk and the master disk.
- Satellite cartridge -- any cartridge which is not the master cartridge. It may be either a systems or non-systems cartridge.

You see, then, that there is a definite distinction between these terms. A disk cartridge is either a systems or non-systems disk, depending on whether you have loaded the Monitor system onto it. On the other hand, the master/satellite split does not occur until the cartridges are placed in the drives, made ready, and a cold start performed. Then, one becomes the master, and the others, if any, become satellites.

The terminology of the disk drives themselves involves another distinction -- that of physical drives versus logical drives. Single-drive 1130 users need not concern themselves with this; their one disk drive is physical drive 0 and logical drive 0 -- there are no options.
- Each disk drive on the 1130 has a physical drive number; drive 0 is the one contained in the mainframe of the 1130; drives 1 through 4 are contained in the 2310 enclosure, a separate unit. These numbers are fixed and cannot be changed.
- Each disk drive present on the 1130 may also be given a logical drive number, which may or may not agree with its physical number. The only
restraint is that a two-drive system may only have physical and logical numbers 0 and 1; a four-drive system, 0, 1, 2, and 3; etc.

You assign logical drive numbers when you prepare a Job card. The Job card may contain a series of five four-digit numbers, representing the ID numbers of each cartridge (each cartridge must be given a four-digit ID when it is initialized). The first of the five ID's (cc 11-14) informs the Monitor that logical drive 0 is to be the drive containing the cartridge with that ID. For example, if this field contained 1234, the drive in which cartridge 1234 is mounted becomes logical drive 0. That cartridge may be physically located on any drive; its actual position does not matter.

Cartridge 1234 would also become the master cartridge, since the cartridge on logical drive 0 will always be the master.

For further detail, see the Monitor reference manual.
DISK STORAGE LAYOUT

Introduction

Conceptually, disk storage can be divided into five logical areas:
- Cylinder 0
- IBM Systems Area
- User Area
- Working Storage
- Fixed Area

The contents and use of these areas are discussed in detail in the Monitor SRL manual, and in general terms here.

Note that these areas are logical or symbolic, rather than physical areas. They are not necessarily intact or contiguous. Some of the items in one logical area may, in fact, be physically located between two items in another logical area.

The term "logical", as it is used here, denotes a system organized for ease of understanding, rather than for accurate technical detail.
Cylinder 0

This area contains certain key information that is present on every disk cartridge. The exact contents of this area differ, depending on whether the disk in question is a systems disk (in which case it contains the Monitor) or a non-systems disk; the area, however, is always present, and always occupies one cylinder, Cylinder 0.
IBM Systems Area

The IBM Systems Area is present on all disk cartridges that have been built as systems disks (that is, disk cartridges on which the Monitor system has been loaded).

This area consists of (1) a basic Monitor package of 152 sectors, which must be present, (2) two optional items, which may be removed:
   FORTRAN compiler (88 sectors)
   Assembler (32 sectors)
and (3) the Core Image Buffer (16 sectors), which may be deleted from a satellite cartridge but must be present on the master cartridge.
Working Storage (WS)

Working Storage is used for temporary storage of programs and data. Since it is used for this purpose by both you and the Monitor, you should not leave material in WS if you wish to use it later. If you wish to retain a program or data file, it should be transferred with DUP to either the User Area or the Fixed Area, and given a name.

The size of WS is variable, since it consists of whatever space on the disk is not taken up by the other four areas.
User Area (UA)

As mentioned earlier, programs and data that you want retained must be moved from WS to either the User Area or the Fixed Area.

The size of the UA is also variable, since it expands and contracts as material is stored in it or deleted from it.

The process of transferring a program or data file from WS to UA is done in a unique manner, made possible by the use of a "floating" boundary between the two areas. Because material placed in WS is at the "lower" end of WS which is adjacent to the "upper" end of UA, all that is necessary to transfer it from WS to UA is to move the boundary. (See Figure 60.1.)

The term "User Area" should not be taken to mean that only user-written programs will be found there. Nearly the entire IBM subroutine library is placed in the UA (occupying about 50 sectors), where it may be called for use by other programs.

The UA may contain:

- Data, in disk data format (DDF)
- Programs and subprograms, in disk system format (DSF)
- Programs, in disk core image format (DCI)

The major differences between these three formats are discussed in subsection 60.30.10.

The Location Equivalence Table (LET) is a directory of the contents of the User Area. It exists on every disk cartridge -- systems and non-systems. Basically, it contains an entry for every program, subprogram, and data file that has been placed in the UA. Each entry in the table contains the name, size, and other properties of that program or data file.
Fixed Area (FX)

The Fixed Area, like the User Area, is a place where the user may store programs and/or data files. There are five major differences between the FX and the UA:

1. There is no Fixed Area on a cartridge unless you specifically define one (see 60.30.20).

2. You specify the size of the FX, whereas the UA expands and contracts as items are added to or deleted from it.

3. Like the UA, the FX may contain both programs and data, but the programs must be in disk core image (DCI) format. They cannot be in disk system format (DSF).

4. Programs or data files stored in the FX may be deleted, but the FX will not be repacked, as is the case with the UA. Once an item is stored somewhere in the FX, it stays in the same location until it is deleted.

5. The directory of the FX is FLET, the Fixed Location Equivalence Table, rather than LET, which is the directory to the UA.
Summary

Figure 60.2 illustrates the five logical disk areas and shows the general properties of each.

<table>
<thead>
<tr>
<th>Logical Area</th>
<th>Sub-Areas</th>
<th>Present?</th>
<th>Approximate Size, Sectors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Systems Disk</td>
</tr>
<tr>
<td>Cylinder 0</td>
<td>Always</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>IBM Systems Area</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic</td>
<td>Only on a systems disk</td>
<td></td>
<td>152</td>
</tr>
<tr>
<td>Core Image Buffer</td>
<td>Can be removed from Non-Sys.</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>FORTRAN Compiler</td>
<td>May be removed</td>
<td></td>
<td>88</td>
</tr>
<tr>
<td>Assembler</td>
<td>May be removed</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>Fixed Area (FX)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLET</td>
<td>Not unless defined by user</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Contents of FX</td>
<td>Not unless defined by user</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fixed by the user when he</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>defines a fixed area</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Area (UA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LET</td>
<td>Always</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Contents of UA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- User data files</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- User programs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- IBM subroutine library</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Always</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Always. As delivered, the</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UA contains the IBM Subroutine Library</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Working Storage (WS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contents of WS</td>
<td>Always</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Varies in size — WS is</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>whatever is left over. Every</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sector added to UA is</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>subtracted from WS, every</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sector deleted from UA is</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>added to WS.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 60.2. The five logical areas of the disk
INCREASING THE AMOUNT OF SPACE AVAILABLE TO THE USER

Introduction

As Figure 60.2 shows, there is another way to look at a disk cartridge. Simply stated, at any point in time, the disk can be split into two portions:

- The portion now being used.
- The portion not now being used and therefore available to you.

If you have a data file that you want to store on a disk, you can ask several pertinent questions:

- How much room do I need?
- How much room do I have?
- How can I make more room, if necessary?

The first question is covered in Section 80; the other two are answered in 60.20.10 and 60.20.20, respectively.
How Much Room Do I Have?

It is quite easy to determine how much room is available on any particular disk cartridge; all you need to do is to run the DUP *DUMPLET job. The last item on the printout will have the name 1DUMY (a dummy entry representing empty space), its size in disk blocks (a disk block is 20 words, or 1/16 of a sector), and its starting address (in disk blocks).

This block of empty space is equivalent to Working Storage, the area where you may place additional programs and data files.

Figure 60.3 shows the last page of a typical DUMPLET printout. Note the last entry:

<table>
<thead>
<tr>
<th>1DUMY</th>
<th>49F3</th>
<th>1AOD</th>
</tr>
</thead>
</table>

Convert the two hexidecimal numbers to decimal:

\[
\begin{align*}
49F3 & \quad \text{becomes} \quad 18931 \\
1AOD & \quad \text{becomes} \quad 6669 \\
\end{align*}
\]

Divide by 16 (16 disk blocks per sector):

\[
\begin{align*}
18931/16 & \quad \text{is} \quad 1183 \ 3/16 \\
6669/16 & \quad \text{is} \quad 416 \ 13/16 \\
\end{align*}
\]

The first number (1183 3/16) is the size in sectors of Working Storage; the second (416 13/16) is the sector at which it begins. The fact that the two add up to 1600, the total number of sectors on a disk, confirms the accuracy of the arithmetic.
How Can I Make More Space Available?

Using Figure 60.2 as your guide, take a look at each of the five logical areas, with an eye toward removing items you don't need:

Cylinder 0

Since Cylinder 0 is always present and necessary on every disk cartridge, there is nothing you can do to reduce its size.

IBM Systems Area

Every system disk cartridge, after initial loading with the Monitor, contains the Assembler and FORTRAN compiler, two programs of substantial size. The Assembler occupies 32 sectors; the FORTRAN compiler occupies 88 sectors.

If you rarely compile programs written in Assembler Language, you will probably want to delete the Assembler from all disk cartridges except the one used for odd jobs.

Most 1130 users program in FORTRAN, but it is still possible to eliminate this compiler from some disk cartridges. Suppose you have a large inventory file that requires all the room you can get. Why keep the FORTRAN compiler on that disk?

During the test phase, when you are compiling many FORTRAN programs, you certainly need the compiler; once the programs have been debugged, however, you can eliminate it and increase the size of your file by 88 sectors. If it becomes necessary to change a program on a particular disk, you can recompile the new version using a disk that does contain the compiler, dump the new program on cards with the DUP, remove the FORTRAN disk, replace it with the inventory (no FORTRAN) disk, and load the new card program with DUP. Because this takes a few minutes, you will probably not want to eliminate the FORTRAN compiler from any disk unless the space is needed.

To delete these two programs from a disk, you must use the DUP *DEFINE function, as shown below

```
I/ UO6
I/ DUP
*DEFINE YOUR ASSAMBLER
```

and/or

```
I/ UO6
I/ DUP
*DEFINE YOUR FORTRAN
```
Fixed Area

Because of the way in which the Fixed Area is handled by the Monitor, you should not define one unless you have a specific purpose in mind for it. Remember that the size (and existence) of the Fixed Area is entirely up to you. If you define a 20-cylinder Fixed Area and use only half of it, the other half is completely wasted; the empty space is not transferred to the UA or WS.

To determine what is in the fixed area, you may run the DUP job:

If it is not full, you may reduce its size (see 60.30.20) accordingly, automatically transferring the released area to Working Storage. If later you wish to place something in FX, you may then increase its size.

User Area/Working Storage

Because the UA and WS interact, they must be considered together. Basically, there is never any room in the User Area -- it is always full. Even if you remove something from it, it is still full, since it is immediately packed, and the free space is transferred to WS.

Your job, therefore, is to remove unneeded items from UA, decreasing its size and thereby increasing the size of WS. The entire contents of WS are, after all, available for transfer back to the UA whenever you have something you wish to store on a permanent basis.

The following sections discuss some items that can be removed from the UA.

I/O Subroutines for Devices Not On Your System.

As mentioned earlier, the Monitor, as delivered and loaded on each disk, is a complete system and includes subroutines for every device that can be installed on an 1130 system -- plotter, paper tape reader, etc. If you do not have a plotter, it makes sense to delete the plotting subroutines. As with the FORTRAN compiler and the Assembler, you must make the decision and do the deleting. The Monitor will not check for the presence or absence of a plotter and delete those subprograms on its own. Although you do specify to the Monitor loader (with the REQ cards) which devices are on your system, the loader does not use this information to selectively load the subroutine library. All subroutines are loaded onto the disk, regardless of your 1130 configuration.

Figure 60.4 illustrates what subroutines can be deleted, and how many sectors can be gained. The subroutines noted can be deleted the same as any other subroutine -- for example:

If you don't have this equipment (or if no program on this disk will use these devices) You may delete these subroutines And gain this number of sectors

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>IBM 1627 Plotter</th>
<th>IBM 1132 Printer</th>
<th>IBM 1403 Printer</th>
<th>IBM 1442 Card Read Punch, Model 6 or 7</th>
<th>IBM 1142 Card Punch, Model 5</th>
<th>IBM 2501 Card Reader</th>
<th>IBM 1134 Paper Tape Reader and/or 1055 Paper Tape Punch</th>
<th>IBM 1123 Optical Mark Page Reader</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOTX</td>
<td>FCHAR</td>
<td>PRINT1</td>
<td>PRINT2</td>
<td>CARD0</td>
<td>PIND0</td>
<td>READO</td>
<td>PAPT1</td>
<td>OMPRF1</td>
</tr>
<tr>
<td>POINT</td>
<td>ECHR</td>
<td>PRNTT2</td>
<td>PRNTT2</td>
<td>CARD1</td>
<td>PIND1</td>
<td>READ1</td>
<td>PAPTN</td>
<td></td>
</tr>
<tr>
<td>XYP1</td>
<td>FCHR3</td>
<td>PRINTZ</td>
<td>PRNZ</td>
<td>CARDO</td>
<td>PIND2</td>
<td>READ2</td>
<td>PAFTZ</td>
<td></td>
</tr>
<tr>
<td>FCHR1</td>
<td>ECHR3</td>
<td>ECHRT3</td>
<td>ECHRT3</td>
<td>PCHT2</td>
<td>PIND3</td>
<td>READ3</td>
<td>PAFT2</td>
<td></td>
</tr>
<tr>
<td>FCHR1</td>
<td>ECHR3</td>
<td>ECHRT3</td>
<td>ECHRT3</td>
<td>PCHT2</td>
<td>PIND3</td>
<td>READ3</td>
<td>PAFT2</td>
<td></td>
</tr>
<tr>
<td>FCHR1</td>
<td>ECHR3</td>
<td>ECHRT3</td>
<td>ECHRT3</td>
<td>PCHT2</td>
<td>PIND3</td>
<td>READ3</td>
<td>PAFT2</td>
<td></td>
</tr>
<tr>
<td>FCHR1</td>
<td>ECHR3</td>
<td>ECHRT3</td>
<td>ECHRT3</td>
<td>PCHT2</td>
<td>PIND3</td>
<td>READ3</td>
<td>PAFT2</td>
<td></td>
</tr>
<tr>
<td>FCHR1</td>
<td>ECHR3</td>
<td>ECHRT3</td>
<td>ECHRT3</td>
<td>PCHT2</td>
<td>PIND3</td>
<td>READ3</td>
<td>PAFT2</td>
<td></td>
</tr>
<tr>
<td>FCHR1</td>
<td>ECHR3</td>
<td>ECHRT3</td>
<td>ECHRT3</td>
<td>PCHT2</td>
<td>PIND3</td>
<td>READ3</td>
<td>PAFT2</td>
<td></td>
</tr>
<tr>
<td>FCHR1</td>
<td>ECHR3</td>
<td>ECHRT3</td>
<td>ECHRT3</td>
<td>PCHT2</td>
<td>PIND3</td>
<td>READ3</td>
<td>PAFT2</td>
<td></td>
</tr>
<tr>
<td>FCHR1</td>
<td>ECHR3</td>
<td>ECHRT3</td>
<td>ECHRT3</td>
<td>PCHT2</td>
<td>PIND3</td>
<td>READ3</td>
<td>PAFT2</td>
<td></td>
</tr>
<tr>
<td>FCHR1</td>
<td>ECHR3</td>
<td>ECHRT3</td>
<td>ECHRT3</td>
<td>PCHT2</td>
<td>PIND3</td>
<td>READ3</td>
<td>PAFT2</td>
<td></td>
</tr>
<tr>
<td>FCHR1</td>
<td>ECHR3</td>
<td>ECHRT3</td>
<td>ECHRT3</td>
<td>PCHT2</td>
<td>PIND3</td>
<td>READ3</td>
<td>PAFT2</td>
<td></td>
</tr>
<tr>
<td>FCHR1</td>
<td>ECHR3</td>
<td>ECHRT3</td>
<td>ECHRT3</td>
<td>PCHT2</td>
<td>PIND3</td>
<td>READ3</td>
<td>PAFT2</td>
<td></td>
</tr>
<tr>
<td>FCHR1</td>
<td>ECHR3</td>
<td>ECHRT3</td>
<td>ECHRT3</td>
<td>PCHT2</td>
<td>PIND3</td>
<td>READ3</td>
<td>PAFT2</td>
<td></td>
</tr>
</tbody>
</table>

Figure 60.4. I/O subroutines which may be deleted.
Computational Subroutines You Are Unlikely To Use. Let's take the example again of the disk used exclusively for a large inventory file. You have eliminated the compilers, the plotter subroutines, etc. Is there anything else on this disk that you won't need? Unless you have an unusual inventory system, the answer is yes. Do the inventory programs require the computation of any sines, cosines, etc? If not, you may gain 7 sectors by deleting the trigonometric and logarithmic subroutines:

FSQR  ESQR
FTANH  ETANH
FATN  EATN
FAXB  EAXB
FEXP  EEXP
FLN  ELN
FSIN  ESINE

Seldom-Used Programs and/or Data. Because the 1130 Monitor makes it so easy to do so, many people tend to "overstore" the disk. This is particularly true of programs, which are often *STOREd as a matter of course, with no rules regarding what gets *STOREd and what doesn't. As a practical matter, however, many programs should not be placed on the disk, but should be compiled each time they are used. For example, suppose that program XYZ is a stand-alone program that does nothing but read a deck of cards and produce one or two pages of results. It is run monthly, consists of 150 FORTRAN source cards, and uses 2100 words of core storage. To compile (without listing) and execute it, will take about:

<table>
<thead>
<tr>
<th>Task</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile</td>
<td>2 minutes</td>
</tr>
<tr>
<td>Execute</td>
<td>3 minutes</td>
</tr>
<tr>
<td>Total</td>
<td>5 minutes</td>
</tr>
</tbody>
</table>

To load it from the disk and execute it, will take about:

<table>
<thead>
<tr>
<th>Task</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>1/2 minutes</td>
</tr>
<tr>
<td>Execute</td>
<td>3 minutes</td>
</tr>
<tr>
<td>Total</td>
<td>3 1/2 minutes</td>
</tr>
</tbody>
</table>

By storing this program on the disk, you will save 1 1/2 minutes per month, but will use 2100 words of disk storage, or about seven sectors.

Is it worth it? That depends on your installation. If disk space is scarce, the answer is: "No -- don't store it!" If there is plenty of room on the disk, the answer is: "Yes, why not?"

Obviously, some programs should or must reside on the disk:
- Often used subroutines and functions
- Programs called as LINKS by other programs
- Frequently used programs
- Large programs
- Programs that are run with a series of other programs, as one batch JOB.

Unneeded User-Written Programs and Data. This usually applies more to programs than data. Over a period of months, the typical disk becomes cluttered with numerous abandoned, obsolete, and/or useless programs and subprograms. The LET/FLET should be dumped periodically and inspected for such items. Anything not really needed should be deleted.
Summary

To illustrate how much room can be available on a systems disk, let's assume you have an 1132 Printer and a 1442 Card Read Punch, and you wish to place a very large commercial-type data file on the disk. There is no Fixed Area.

After originally loading the Monitor, you *DUMPLET and determine from the last IDUMY record that the size of Working Storage is 49F3 disk blocks, or about 1183 sectors, 74% of the disk.

To increase this amount, you can take the three steps suggested earlier:

1. Delete the FORTRAN compiler and the Assembler, gaining 120 sectors.
2. Delete the I/O subroutines you don't need, in this case gaining about 37 1/2 sectors.
3. Delete the technically oriented computational subprograms, gaining about seven sectors.

You thereby have increased the available disk space (WS) by 164 sectors, to 1347, or 84% of the disk. Of course, you cannot compile any programs with this disk, nor can you execute any jobs (noncommercial) requiring some of the computational subroutines that have been deleted. From the number of sectors available you must subtract the space required for your programs. The remainder is available for your data file(s).

The task is easier with a non-systems disk. One cylinder (eight sectors) is always required for the Cylinder 0 area, plus two more if you have defined a Fixed Area. That leaves either 1584 or 1576 sectors for your programs and data files.
THE DISK UTILITY PROGRAM

Introduction

The Disk Utility Program (DUP) gives you the facilities necessary to manage your disk storage capability. With DUP you can:

- Store programs and data files on the disk
- Make the programs and data files on the disk available in printed, punched card, or punched paper tape form
- Remove programs and data files from the disk
- Determine the contents of disk storage through a printed copy of LET/FLET, the directory to the disk
- Alter certain system parameters and, to a limited extent, the contents of the system
- Perform other minor disk maintenance functions

The Monitor manual explains the details required to use DUP (card layouts, etc.). This section will cover only the most commonly required DUP functions and the information needed to execute them.
Format of Material on the Disk

Essential to the understanding of DUP is a basic knowledge of the various formats used in the storing of programs and data on the disk.

Although DUP gives you many format options, this section discusses only those that apply to the average user, writing a typical FORTRAN program. Users with unusual combinations (for example, a data file in DCI format) will have exercised this option with a specific purpose in mind and will be well aware of the details involved.

Data Files

Under normal circumstances, data files are always stored on the disk in the Disk Data Format (DDF).

Programs and Subprograms

Under normal circumstances, programs and subprograms will be stored on the disk in one of two formats:

- Disk System Format (DSF)
- Disk Core Image Format (DCI)

The main difference between the two lies in what is stored, rather than how it is stored.

A program in DCI format consists of a complete, self-sufficient core load or program package -- the mainline program, plus all the subroutines it requires. The entire package is in absolute form; that is, all addresses are actual core storage locations rather than relative locations. Subprograms cannot be in DCI format.

On the other hand, an item in DSF consists of that item and only that item. Nothing else is included with it. It may be:

- A program or a subprogram
- Absolute or relocatable (but usually relocatable)
- In either WS or UA (but not in the FX)

As would be expected, a program occupies more space on the disk in DCI form than it would in DSF, since it includes more material. However, it may be loaded into core storage (when called by an XEQ card) much faster, since the Core Load Builder need not assemble all the necessary subroutines and calculate actual core storage addresses.
The Most Commonly Used DUP Functions - Single Disk Drive Systems

Of the many things that can be done with DUP, a few stand out as common, everyday tasks in the typical 1130 installation. The following is a guide to these common jobs:

Store a Program or Subprogram in DSF Format

After compiling a program or subprogram, you will commonly store it on the disk for later reference or execution.

Because the FORTRAN Compiler (or Assembler) leaves the compiled program in Working Storage, all that need be done is to move it from WS to UA. To do this, DUP moves the boundary between UA and WS so as to include in UA whatever is in WS. For example, suppose you have just compiled a program called PROGZ, which requires 812 words of core storage. If you follow the END card of the program with

```
1 DUP
2 STORES WS UA PROGZ
```

DUP will move this program (move the boundary) from WS to UA, and enter the name PROGZ in LET, with the proper identification codes. UA increases in size by about 812 words; WS decreases by the same amount. Note that you did not have to know how large the program was --- DUP handles that. Note also that the DUP card is not preceded by a JOB card.
Store a Program in DCI (Core Image) Format

You can, after compilation, also store a program in DCI format, by simply using the *STORECI card in place of the *STORE card. Note that the *FILES, *LOCAL, and *NOCAL cards are placed after the *STORECI card and that the number of such cards is punched in columns 27-30 of the *STORECI card.

Because this takes longer than the *STORE option, it usually will not be done unless you are fairly certain that the program is free of bugs.

Convert a DSF Program to DCI

For speed of loading, commonly used programs should be stored in DCI (core image) format. This eliminates the need to build a core load each time you execute the program.

If you have a program called MAIN6 stored on the disk in DSF (by a STORE card), you can convert it to DCI with the following sequence:

```
/ /  
/ / DUMP
/ / STORECI          US V4 MAIN6 6
/ / LOCAL
/ /  
/ / DELETE
/ / MAIN6
```

Note that the name of the program had to be changed.

Delete a Program or Subprogram

This is one of the simplest of the DUP jobs, since you need not be concerned with either the format, the location, or the type (program, subprogram, or data) of the item to be deleted.

The sequence of cards

```
/ /  
/ / DELETE
/ / NAMEP
```

will delete NAMEP wherever and whatever it is.
Dump a DSF Program or Subprogram and Reload It

As a backup procedure, you can dump your often used programs and subprograms onto cards or paper tape. If anything happened to the disk cartridge, these items could be reloaded much faster than they could be recompiled. The job

```
    01 USE
    02 DUP
    03 DUMP      IA  CA  ITEM
      output cards: about one inch
```

will cause ITEM to be punched into the deck of blank cards following the *DUMP card, 54 words per card. In addition, a header and end-of-program card will be punched.

Since the program is punched in such a compact form, very few programs will require more than an inch of cards (about 140 cards, or 6300 words). Extra, unpunched cards will be bypassed automatically by DUP.

To reload this dumped program, the *DUMP card should be replaced with

```
    01 USE
    02 DUP
    03 STORE      CA  IA  ITEM
```

(if the program was in DSF) and run as another job.

Dump a DCI (Core Image) Program and Reload It

If the program to be dumped and reloaded is in core image format, the procedure must be changed somewhat.

The dump to cards can be accomplished in the same way, with the *DUMP card.

However, to reload, the STOREDATACI option is required, and the card count must appear in columns 27–30. For example, a program called XXXXX, dumped into a deck of 108 cards, would be reloaded with the card:

```
    01 USE
    02 DUP
    03 STORE      CA  IA  ITEM
      output cards: about one inch
```

will cause ITEM to be punched into the deck of blank cards following the *DUMP card, 54 words per card. In addition, a header and end-of-program card will be punched.

Since the program is punched in such a compact form, very few programs will require more than an inch of cards (about 140 cards, or 6300 words). Extra, unpunched cards will be bypassed automatically by DUP.

To reload this dumped program, the *DUMP card should be replaced with

```
    01 USE
    02 DUP
    03 STORE      CA  IA  ITEM
```

(if the program was in DSF) and run as another job.
Dump a Data File and Reload it

More important as a backup procedure, you can dump your data files onto cards or paper tape. In case anything happens to the disk cartridge, the data file may be reloaded.

To dump a data file, you must know its size in sectors.

The sequence of cards

```
11 JOB
11 DUP
STOREDATA UA CO FILEX 0365
enough blank cards to fill file
```

will dump the 65-sector data file, FILEX, from the UA to cards (CD).

The data file is punched into the blank card deck, 54 words per card. No header or trailer cards are punched.

To reload, you must know the number of cards in the dumped deck.

To continue the previous example of a 65-sector file (20,800 words), the dumped deck would have required about 386 cards.

To reload, then, you need the cards

```
11 JOB
11 DUP
STOREDATA CO UA FILEX 0365
the dumped data file deck
```

Copy a Data File onto Another Area on the Same Disk

Another method of data file backup is to copy the file onto another portion of the disk. Typically this would be done before running a job that modifies the file. If the file is 100 sectors long and called MEN, the job

```
11 JOB
11 DOP
STOREDATA UA WS MEN 0100
STOREDATA WS UA TMEN 0100
```

will move it to Working Storage, then include it in the User Area with a different name (TMEN).

If the program you wish to run operates satisfactorily, updating the file MEN, you need do nothing except DELETE TMEN.

If on the other hand, some error occurs that ruins the file MEN, you have a duplicate file (TMEN) ready to replace it. The steps shown below will replace MEN, which has been ruined, with TMEN:

```
11 JOB
11 DOP
DELETE UA MEN 0100
STOREDATA WS UA TMEN 0100
```

Now you may rerun the job. Be especially careful not to *DELETE TMEN until you are sure everything went according to plan.

This protects you from accidental programmed loss of a data file; however, it does not protect against physical loss or destruction of the disk cartridge itself.
Defining and Modifying the Fixed Area

If you want a Fixed Area on a disk cartridge, you must not only instruct the monitor to create one, but you must specify its size.

If you want a Fixed Area of 20 cylinders, you can run the job

```
II JOB
II DUP
DEFINE FIXED AREA    0001
```

and you have it. Note that we specified 21 cylinders as the size of the Fixed Area. One cylinder will be used for FLET; the other 20 are available for your programs and data files.

If later you wish to increase the size of FX by 6 cylinders, you can use

```
II JOB
II DUP
DEFINE FIXED AREA    0006
```

or, if you wish to decrease its size by 3 cylinders

```
II JOB
II DUP
DEFINE FIXED AREA    0005
```

You should keep a record of whether a particular cartridge has a Fixed Area or not. If you ran the first job, then forgot you ran it, and ran it again, you would have a 41-cylinder Fixed Area. When in doubt you may use the DUMPLET DUP option, which will print the contents of FLET.
Special Options -- Multiple Disk 1130 Users

Copy a Data File onto Another Disk

If you have more than one disk drive, you will usually take this option rather than the ones described earlier -- dump to cards for backup, or copy to another part of the same disk. This requires a two-step procedure, since data files cannot be copied directly from the UA on one disk into the UA on another disk. The transfer must be via WS.

Suppose you have an 88-sector file, called DATAX, in UA on cartridge 1075, and you want to copy it into the UA on cartridge 1077. Assume that cartridge 1075 is on drive 0, and 1077 is on drive 1. The following card sequence will accomplish this task:

This copies the program or subprogram called PROGX from cartridge 1075 to cartridge 1077. As before, the program now exists on both cartridges, each of which has its own LET.

Neither the format (DSF or DCI), nor the type (program or subprogram) need be known, or specified.

Copy an Entire Disk onto Another Disk

This is not done with the Disk Utility Program (DUP) but with a Disk Maintenance Program called COPY, which is supplied with the Monitor. If you want to copy the entire contents of cartridge 1967 onto cartridge 1968, you execute COPY:

Copy a Program onto Another Disk

If you have multiple disks, you may also choose to back up your programs by copying them to another disk, rather than dumping to cards. This is similar to the previous task, but easier, since you do not have to know the size of the program, as was the case with a data file. You must still, however, go via WS in the two-step procedure:
Section 65: THE MONITOR - CORE STORAGE MANAGEMENT

CONTENTS

Introduction ............................................. 65.01.00
The Logical Layout of Core Storage .... 65.10.00
Basic ...................................................... 65.10.10
Flipper ...................................................... 65.10.20
SOCAL Area .............................................. 65.10.30
General
Overlay 1
Overlay 2
Overlay 3
The SOCAL Overlay Scheme
Possible Improvements to the SOCAL Scheme

Reduce the Size of the Largest SOCAL Overlay
Combine Overlays 1 and 3
LOCAL Area ............................................ 65.10.40
General
IBM-Supplied (Systems) Subroutines
Program or LINK Area ......................... 65.10.50
COMMON Area ..................................... 65.10.60
Unused Area ........................................ 65.10.70
Summary .............................................. 65.20.00
INTRODUCTION

The 1130 Disk Monitor System gives you three extremely powerful and useful means of managing core storage. All three involve the sharing of core storage by two or more programs (LINKs), subprograms (LOCALs), or groups of subprograms (SOCALs). This section describes these three schemes in detail, after discussing the 1130 core storage layout in terms of its seven logical areas.
THE LOGICAL LAYOUT OF CORE STORAGE

You can think of core storage as consisting, like the disk cartridge, of several logical areas. Again, this layout may bear little or no resemblance to the actual, physical layout; it is merely a device to help you understand the dynamic nature of core storage.

The seven logical areas are as follows:

- Basic
- Flipper
- SOCAL Area
- LOCAL Area
- Program Or LINK Area
- COMMON
- Unused

These areas are described below in general terms. Complete details may be found in the appropriate Monitor reference manual. Note that all core sizes given are based on:

1. A typical FORTRAN program--commercially rather than scientifically oriented.
2. Approximate subroutine sizes, usually adjusted to multiples of 10.
3. Version 2, Modification Level 0, of the 1130 Disk Monitor System.

Because some of the package sizes may increase in the future, you should not plan on using all of the available core storage; it might be more prudent to use about 95% of it.
**Basic**

This is a set of programs that is always in core and whose size varies only slightly from job to job. It consists of:

1. Resident Monitor
2. Transfer Vector
3. Several commonly used subroutines kept in core storage at all times (IFIX, FLOAT, ELD, ESTO, NORM, etc.). These are all subprogram subtypes 0 -- see discussion of subtype under "SOCAL Area".

A good average size for this area is 740 words.
Flipper

This routine handles both the SOCAL and LOCAL overlay system. Flipper is not required (core size = 0) if there are no SOCALs or LOCALs: if there are, its size is about 100 words.
The word SOCAL is an acronym derived from "System Overlay on Call". The SOCAL area is that area of core storage where the SOCAL subroutines reside. The SOCAL subroutines, in turn, are defined as those subprograms that:

1. Are used by the mainline program to be executed.
2. Have been designated as subtype 1, 2, 3, or 8.
3. Have not been made LOCAL.

If a subprogram has not been designated as subtype 1, 2, 3, or 8, it will be located in one of three areas:

1. The LOCAL area if it has been specified as LOCAL.
2. The Basic area if it is an IBM-supplied subprogram (IFIX, FLOAT, ELD, EST, etc.) and has not been made a LOCAL.
3. The Program area if it is a user-supplied subprogram and has not been made a LOCAL.

The 1130 Monitor system you receive from IBM includes a subroutine library in which each subroutine is assigned a subtype number. These may be called the standard subtypes, and will yield a SOCAL system as described in the Monitor manual and in later subsections of this Guide. However, these subtype numbers may be changed at your discretion. Furthermore, you may assign subtype numbers to your own subprograms. Both steps will yield a nonstandard SOCAL system. Several ideas on this subject are presented later in this subsection.

The SOCAL system involves the grouping of the SOCAL subroutines into three groups, called overlays, which will be manipulated by the Core Load Builder as it goes about its job of loading your program into core storage.

Overlay 1. This is made up of all those subroutines and functions designated as subtype 2 or 8. The ARITHMETIC, PAUSE, and STOP routines are subtype 2; the functionals (SIN, COS, etc.) are subtype 8.

The "typical" commercial program will probably add, subtract, multiply and divide (in extended precision), PAUSE, STOP, and read the data switches. The subroutines required to do this will occupy about 520 words of core storage. If the program does not divide, the size of this overlay will be reduced by 180 words.

Commercially oriented 1130 programs will probably be limited to these subroutines, while technical-type jobs may use the SIN, COS, SQRT, etc., functions and require up to several hundred more words.
Overlay 2. Overlay 2 is composed of all subtype 3 subroutines—those required for non-disk input/output. The basic component is SFIO, the Format Interpreter, which is required if the program to be executed contains any non-disk FORTRAN I/O statements. In addition, each I/O device requires its own I/O subroutine and often several code conversion routines.

The size of this overlay varies considerably, depending on the I/O devices specified on the *IOCS card (whether they are used or not). The following table may be used to calculate the approximate size of this overlay.

<table>
<thead>
<tr>
<th>If your program contains any:</th>
<th>This many words will be included in overlay 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Non-disk formatted input/output (SFIO)</td>
<td>1150</td>
</tr>
<tr>
<td>b) WRITE on the 1132</td>
<td>190</td>
</tr>
<tr>
<td>c) WRITE on the 1403</td>
<td>190</td>
</tr>
<tr>
<td>d) WRITE on the 1442-5</td>
<td>70</td>
</tr>
<tr>
<td>e) WRITE on the console printer (typewriter)</td>
<td>60</td>
</tr>
<tr>
<td>f) READ or WRITE on the 1442-6 or 7</td>
<td>160</td>
</tr>
<tr>
<td>g) READ from the 2501</td>
<td>60</td>
</tr>
<tr>
<td>h) READ from the keyboard (cannot be done without writing on console printer)</td>
<td>30</td>
</tr>
<tr>
<td>i) READ from keyboard or 2501 or 1442-6, 7</td>
<td>190</td>
</tr>
<tr>
<td>j) READ or WRITE on paper tape</td>
<td>225</td>
</tr>
</tbody>
</table>

Consider, for example, a FORTRAN program compiled with the card:
*IOCS (1132 PRINTER, TYPEWRITER, KEYBOARD)
Referring to the table above, this program will require the following:

<table>
<thead>
<tr>
<th>Item</th>
<th>Reason</th>
<th>No. of Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>There will be formatted I/O using non-disk units.</td>
<td>1150</td>
</tr>
<tr>
<td>b</td>
<td>The 1132 printer is specified.</td>
<td>190</td>
</tr>
<tr>
<td>c</td>
<td>The typewriter is mentioned.</td>
<td>60</td>
</tr>
<tr>
<td>d</td>
<td>The 1442 is included.</td>
<td>160</td>
</tr>
<tr>
<td>i</td>
<td>The program READs from the 1442.</td>
<td>190</td>
</tr>
</tbody>
</table>

This program, therefore, will require a 1750-word overlay. (Note again that it is the *IOCS card, not your program, that determines the size of this package.)
Overlay 3. This is the FORTRAN disk I/O package, which may contain:

- SDFIO (620 words), the disk I/O package
- SDFND (80 words), the disk FIND package
- SUFIO (730 words), the disk unformatted I/O package

All three subroutines are subtype 1. The size of this package, therefore, ranges from 0 (no disk I/O) to 1430 words.

Note that SDFND is not included unless your FORTRAN program contains a FIND statement. SDFIO is included if the *IOCS (DISK) card is present; SUFIO if the *IOCS (UDISK) card is present.

The typical program will require SDFIO and SDFND, for an overlay size of 700 words.

The SOCAL Overlay Scheme

Just before you execute a program or store one in core image format (DCI), the Core Load Builder (CLB) is given the task of building a complete core load, or program package, which will fit into core storage.

CLB assembles your program and all its required subroutines, and determines how much core storage they will require. In so doing, it considers the subroutines that are to be LOCAL. The CLB then tries to include the last remaining elements, the three SOCAL overlays, in four steps:

1. As a first step, CLB attempts to fit all three overlays in core with no sharing. Using the typical overlay sizes, this will require 520 + 1750 + 700 or 2970 words of core.

2. A second step is taken if there is not enough room to hold all three packages at the same time. This involves the sharing of core storage by overlay 1 (arithmetic) and overlay 2 (non-disk I/O). The area they share must be large enough for the larger of the two overlays, in this case (and almost always) the non-disk I/O subroutines, overlay 2. The size of the SOCAL area will now be 1750 + 700 or 2450 words, a reduction of 520 words, the size of overlay 1.

As required by the user's program, Flipper will read each overlay from the disk whenever it is needed, placing it on top of the last overlay. Overlay 3, the disk I/O, will remain in core at all times. Because Flipper is now needed, your net gain is 520 - 100 or 420 words.

3. The third step is taken if there is still not enough room in core storage. It involves the sharing of core storage by all three packages, in an area the size of the largest of the 3 overlays. As before, this will probably be the non-disk I/O overlay, at 1750 words.

4. If step 3 fails to provide enough room in core, step 4 will so advise you with a message.

Summarizing the CLB makes a step-by-step attempt to fit your program and its subprograms into the available core storage space.

- Step 1 involves the most core storage -- typically about 2970 words.
- Step 2 requires about 520 - 100 or 420 words less than step 1.
- Step 3 requires about 700 words less than step 2.

Figure 65.1 shows the three steps, or overlay levels, in graphic form. Note that the discussion of this typical program did not include the program itself. Only the subprograms have been considered.
If you place an L in column 14 of the // XEQ card, the Core Load Builder will print a core map showing which subprograms, if any, are in which SOCAL overlay, and the size of each overlay. (See Figure 65.5 for such a map.)

Possible Improvements to the SOCAL Scheme

Figure 65.1 illustrates, to a rough scale, the layout of the SOCAL area at each overlay level. One fact is apparent: overlay 2 is much larger than either overlay 1 or overlay 3, and is, in fact, larger than the two combined. Since the SOCAL area must be at least as large as the largest of the three overlays, a certain amount of core storage is unused in some circumstances.

On the basis of this fact, there are two techniques that may be used to make the standard SOCAL system more effective:

Reduce the size of the largest SOCAL overlay. Since LOCALs, discussed later, take precedence over SOCALs, you have a means to remove subprograms from the SOCAL area and to force them into the LOCAL area. Naturally, you would do this only to subprograms in the largest overlay, usually the non-disk I/O package.

Because one LOCAL cannot call another LOCAL, you must be somewhat careful here. For example, you cannot LOCALize both the 1132 subroutine and a subroutine that calls it. One or the other may be LOCAL, not both.

If you are sure such a situation does not exist, you can make the following subroutines LOCAL:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required for</th>
<th>Size in Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARDZ</td>
<td>1442 Card Read Punch</td>
<td>160</td>
</tr>
<tr>
<td>PNCHZ</td>
<td>1442-5 Card Punch</td>
<td>70</td>
</tr>
<tr>
<td>READZ</td>
<td>2501 Card Reader</td>
<td>60</td>
</tr>
<tr>
<td>TYPEZ</td>
<td>Console Printer</td>
<td>60</td>
</tr>
<tr>
<td>WRTYZ</td>
<td>Console Keyboard and Printer</td>
<td>90</td>
</tr>
<tr>
<td>PRNTZ</td>
<td>1132 Printer</td>
<td>190</td>
</tr>
<tr>
<td>PRNZ</td>
<td>1403 Printer</td>
<td>190</td>
</tr>
<tr>
<td>PAPTZ</td>
<td>Paper Tape Units</td>
<td>225</td>
</tr>
</tbody>
</table>

(If you accidentally do make one LOCAL call another LOCAL, the LOADER will call it to your attention with an error message.)

Each of these routines, if made LOCAL, releases as much core storage as the size of the routine. It is unlikely, however, that you can reduce overlay 2 to the same size as the other two overlays unless you LOCALize the entire 1150-word Format Interpreter (SFI0).
To see what that would do to the SOCAL system, let us observe what the three overlays would be if SFIO were LOCAL (and therefore not SOCAL):

Overlay 1 ARITHMETIC (about 520)
Overlay 2 CARDZ, PRNTZ, TYPEZ, etc. (about 600)
Overlay 3 DISK I/O (about 700)

You have not saved the entire 1150 words of SFIO, because now your disk I/O package, overlay 3, at 700 words, is the largest. Your net gain in the SOCAL area is 1750-700 or 1050 words of core storage. Furthermore, the LOCAL SFIO at 1150 may now be the largest of the LOCALs, consequently enlarging your LOCAL area; so you may not really have saved 1050 words. If the largest LOCAL previously was 800 words in length, and the LOCAL area is now 1150-800, or 350, words larger, your net gain is 1050-350 or 700 words. This is still substantial.

Because all READs and WRITEs (except to the disk) use SFIO, making SFIO LOCAL rules out the possibility of making LOCAL any subroutine containing non-disk I/O. This may hamper your flexibility in using LOCALs and further reduce your 700-word saving.

Combine Overlays 1 and 3. Again observing Figure 65.1, you see that overlay 2 is larger than overlays 1 and 3 together (1750 is greater than 520+700). Why not, therefore, combine these two overlays into one? This will not save any core, but it may reduce the amount of time spent in overlaying one package with another.

Since the subprograms in overlay 1 are all subtypes 2 and 8, and those in overlay 3 are all subtype 1, you need only change SDFIO, SDFND, and SUFIO from subtype 1 to subtype 2, and they will be included automatically in overlay 1.

To do this, you may *DUMP SDFIO, SDFND and SUFIO from the User Area to cards, *DELETE them, then reload the cards with a 2 punched in column 11 of the *STORE cards.

If your programs run more slowly or no longer fit in core, *DELETE the subtype 2 routines and reload the card decks, this time with a 1 in column 11 of the *STORE card. This will restore them to their original state.

Figure 65.2 illustrates how your SOCAL area is affected by this change. For the typical program, overlay 2 remains at 1750 and overlay 1 grows to 520+700 or 1220 words. Since there are no longer any subtype 1 subroutines, overlay 3 will have a size of zero words, and the CLB will, in effect, skip step 3.
Local Area

General

The LOCAL (LOad-on-CAL1) area is a second area in core storage where the Monitor will overlay subprograms, although in a manner different from the SOCAL scheme in these respects:

1. You must specifically designate a subprogram as LOCAL. It is not automatic.
2. These subprograms are not grouped by any subtype. Each subprogram forms one overlay, and each overlay contains one subprogram.
3. You are not limited to three overlays. If you have 17 subprograms, you may make all of them LOCAL, thus creating 17 LOCAL overlays.

Like the SOCAL area, the LOCAL area will be as large as the largest LOCAL subroutine.

LOCALs and SOCALs do not overlay one another. There are two areas in core storage for subprogram overlays—one as large as the largest SOCAL overlay and another as large as the largest LOCAL subprogram.

To give some examples of how LOCALs are used, take a program that uses five functions and/or subroutines, called SUB1, SUB2, SUB3, SUB4, and SUB5. You may designate none, one, two, three, four, or all five as LOCAL. Those that are LOCAL will overlay one another, being read from the disk whenever required; those that are not LOCAL will remain in core storage at all times.

Subroutines must be specified as LOCAL, with the *LOCAL card, every time a DSF program is executed, or at the time a core load is built with a *STORECI card. Suppose you have a main program XXXX, which uses the five subprograms mentioned above:

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB1</td>
<td>300</td>
</tr>
<tr>
<td>SUB2</td>
<td>60</td>
</tr>
<tr>
<td>SUB3</td>
<td>378</td>
</tr>
<tr>
<td>SUB4</td>
<td>406</td>
</tr>
<tr>
<td>SUB5</td>
<td>19</td>
</tr>
</tbody>
</table>

Total 1163 words

If you execute XXXX with the cards

you will reduce your core storage requirements by 1153–406 or 747 words, since only enough room for the largest, SUB4, at 406 words, is needed, rather than enough for all five, 1153 words.

If you execute XXXX with the cards

you will reduce your core requirements by the size of SUB3 (378 words), since it and SUB4 will overlay each other. SUB1, SUB2, and SUB5 will be in core all the time, since they are not mentioned on your *LOCAL card.

There are several other options in the preparation of the *LOCAL card. For example, the above example could also have been
where the comma after SUB3 implies continuation, or

```
// JOB x x x K
// L C 3
```

IBM-Supplied (Systems) Subroutines

In addition to your own subprograms, you may also designate many of the IBM-supplied subprograms as LOCALs. All subroutines and functions except ILS00, ILS01, ILS02, ILS03, and ILS04, the Interrupt Level subroutines, can be made LOCAL. As a practical matter, however, it is often difficult to LOCALize such subroutines, because many of them call several other subroutines, and one LOCAL cannot call another LOCAL.

This was mentioned earlier, when it was suggested that some subprograms, ordinarily SOCALs, could in fact be made LOCAL instead.

If the program to be executed has just been compiled, it is located in Working Storage and therefore has no name. The *LOCAL card in this case would appear as

```
// JOB
// KED
// LOCAL XXXX, SUB77
```

without a name for the Mainline (calling) program. (Note the comma in its place.)

If program XXXX calls program ZZZZ as a LINK (CALL LINK (ZZZZ)), you must specify the LOCALs for ZZZZ also, at the time you tell the Monitor to execute (or *STORECI) XXXX

```
// JOB
// KED XXXX
// LOCAL XXXX, SUB77, SUB91
```

where SUB77 and SUB91 are other subroutines LOCAL to ZZZZ.
Program or LINK Area

This area will contain
1. Your mainline program
2. All of your subprograms that are not LOCAL or SOCAL.
3. All of the IBM-supplied subprograms that are not LOCAL, SOCAL, or subtype 0.
4. All data (variables and constants) used by the mainline and/or its subprograms, not placed in COMMON.

This forms the third area in core where overlays may be employed; in this case one program package, or LINK, will overlay another.

As in the case of LOCALs, this is not done automatically; it must be planned and executed by you.

Suppose you have written a very large (10,000-word) program, named BIG. When you try to execute it, you are informed by the Monitor that it is too big. Looking at the program, however, you see that it can actually be thought of as four programs, connected as shown in Figure 65.3.

If you split BIG into four programs and place the CALL LINK statements in the proper places, the four will run essentially the same as one large program (although possibly a little slower). Each program or LINK may have its own SOCALS, LOCALs, variable data, subprograms, etc. However, if the LINKs must communicate with each other through core-resident data (rather than disk data), this data must be placed in the COMMON area, with the COMMON statement (see next subsection). During execution of such a program, while the location and contents of the SOCAL, LOCAL, and LINK areas may be continually changing, the COMMON area does not change. It stays in the same place and is not involved in any overlay.

Figure 65.3. A program, "BIG", segmented into four links
There are many different ways you can accomplish this, the easiest being to compose one COMMON statement

COMMON DATE, TABLE(100), K, X, Y, ANS

and include it in BIG1, BIG2, BIG3, and BIG4.

Another way would be to use the following COMMON statements:

in BIG1 COMMON DATE, TABLE(100)  
in BIG2 COMMON DATE, TABLE(100), K, X, Y,  
in BIG3 COMMON DATE, TABLE(100), K, X, Y, ANS  
in BIG4 COMMON DATE, TABLE(100), K, X, Y, ANSWR

Here you see that the size of COMMON in BIG1 and BIG2 is reduced, since unneeded items are not retained. Some unneeded items (like K in BIG3) cannot be eliminated, since you must preserve the relative location (structure) of COMMON from one program to the next, not just the name.

Note that the name of the last variable changes from ANS to ANSWR in LINKing from BIG3 to BIG4. This does not matter, since only the relative position in core storage is important, not the name.

There are many other ways in which COMMON may be arranged. To take advantage of the fact that BIG4 does not use X, Y, or the TABLE array, we may use

in BIG1 COMMON DATE, K, ANS, TABLE(100), X, Y  
in BIG2 COMMON DATE, K, ANS, TABLE(100), X, Y  
in BIG3 COMMON DATE, K, ANS, TABLE(100)X, Y  
in BIG4 COMMON DATE, K, ANSWR

which reduces the core requirements of BIG4 by 102x3 (or 2) words, depending on the precision used.
UNUSED Area

This is whatever core storage remains after the other six areas have been loaded. It must be zero or more words in length. Good programming practice suggests that it should be at least 100 words, to provide for future growth of the Monitor System, IBM subroutines, and/or your programs.
SUMMARY

This section has described the seven logical areas of core storage, with the emphasis on their overall roles rather than on exact details. As mentioned earlier, all quoted subroutine sizes are approximate, and are based on a so-called "typical" commercial-type program, coded in FORTRAN. You should not necessarily conclude that these figures will apply to your "typical" programs; they may or may not.

The bulk of the material in this chapter concerns SOCALS, LOCALS, and LINKs--how they work. Section 90 concerns how they should be used and how they affect program performance.

Figure 65.4 graphically summarizes what has been covered in this chapter.

Figure 65.5 shows a "core map", printed if you punch an L in column 14 of the // XEQ card. From this printout you can determine the exact sizes of some of these packages:

- The size of the Unused area is contained in the R41 message.
- The size of the SOCAL area can be determined from the largest value contained in the R43, R44, and/or R45 message.
- The size of the LOCAL area may also be determined from the core map. If SOCALS are present, the size of the LOCAL area is the address of the lowest SOCAL subroutine, less the address of the next higher non-LOCAL. In this case it would be 170C - 1567, or, in decimal, 5900-5479 or 321 words.
- Flipper (FLIPR), if present, is always about 100 words in length.
- The sizes of the other areas--Basic, Program, and COMMON--cannot easily be determined from the load map.
## Section 70: 1130 FORTRAN AND THE COMMERCIAL SUBROUTINES

### CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Subsections</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>00</td>
<td>01</td>
</tr>
</tbody>
</table>

**Introduction** .................................. 70.01.00

**Arithmetic Considerations** ................. 70.10.00

**General** ...................................... 70.10.01

**Integer Mode** .................................. 70.10.10

**Real Mode** ...................................... 70.10.20

**General** ...................................... 70.10.20

- Real -- Floating Point
- Real -- Fixed Point
- Rounding
- Accuracy and Magnitude
- Output of Large Real Numbers
- Multiplication of Large Real Numbers

**Decimal Mode** .................................. 70.10.30

**Introduction** .................................. 70.10.30

**General Principles** .......................... 70.10.30

**The Decimal Arithmetic Subroutines** ....... 70.10.30

- Addition
- Subtraction
- Multiplication
- Division
- Constants
- Testing and Modifying Signs
- Moving Signs
- Comparing Decimal Fields

**Summary** ...................................... 70.10.40

**Overlapped Input/Output** ..................... 70.20.00

**Introduction** .................................. 70.20.00

**The Commercial Subroutine Package** ....... 70.20.01

**Overlapped I/O Subroutines** .................. 70.20.10

**General** ...................................... 70.20.10

- Read a Card, 1442-6 or 7
- Punch a Card, 1442-6 or 7
- Select Stacker, 1442-6 or 7
- Print on 1132
- Skip on 1132
- Type on Console Printer
- Accept Data from Console Keyboard
- A precaution -- IOND

**Using the Overlapped I/O System** ............ 70.20.20

**General** ...................................... 70.20.20

**Overlapping and Your Program** ............... 70.20.20

**FORTRAN TRACE Not Permitted** ............... 70.20.20

**Alphabetic Headings** ......................... 70.20.20

**The Interaction of Arithmetic and I/O** ....... 70.30.00

**Character Handling Techniques** .............. 70.40.00

**General** ...................................... 70.40.01

**Code Conversion** .............................. 70.40.10

- Integer to Real -- FLOAT
- Real to Integer -- IFIX
- A1 to Real -- GET
- A1 to Integer
- Real to A1 -- PUT
- Integer to A1
- A1 to Decimal -- A1DEC
- Decimal to A1 -- DECA1
- A1 to A2 -- PACK
- A2 to A1 -- UNPAC
- Other Code Conversions

**Other Character Handling Techniques** ....... 70.40.20

**Editing Output -- EDIT** ....................... 70.40.20

- Moving Data Fields -- MOVE
- Filling a Field with a Specific Character -- FILL
- Comparing Alpha Fields -- NCOMP
- Match/No Match Alpha Compare
- High/Low/Equal Alpha Compare
- Working with Zone Punches -- NZONE

**The NZONE Subroutine** ....................... 70.40.20

**FORTRAN Core Saving Tips** ................. 70.50.00

**General** ...................................... 70.50.01

**Reducing Program Size** ....................... 70.50.10

- Use the DATA Statement
- Keep FORMAT Statements Compact
- Code Efficient I/O Statements
- Avoid Long Subroutine Argument Lists
- Avoid Arithmetic with Variables
- Having Constant Subscripts

**Reducing Subroutine Requirements** .......... 70.50.20

**Raising a Real Number to a Whole Power** .... 70.50.20

- SQRT vs **.5
- Don't Include Unneeded I/O Devices on *IOCS Card
- Remove FIND Statements If You Have SOCAL's or LOCAL's
- Remove the TRACE from Production Status Programs

**FORTRAN Execution Times** .................... 70.60.00

**Processing** ................................... 70.60.10

**Summary and Conclusion** ..................... 70.60.20
INTRODUCTION

The primary purpose of this chapter is to discuss the use of 1130 FORTRAN in a commercial environment. Many of the topics, however, will also be of use to the technically oriented user. Topics include:

- Arithmetic considerations -- a discussion of integer, real, and decimal arithmetic, with particular attention to the accuracy and magnitude of numerical values
- Input/output--explaining the overlapped I/O subroutines and how they can improve performance
- The interaction between input/output and arithmetic
- Core storage saving tips for FORTRAN programmers
- Estimating run time of FORTRAN programs
ARITHMETIC CONSIDERATIONS

General

Of prime interest to commercial 1130 users is the precision and accuracy of their arithmetic calculations. Many engineering and scientific applications have very little need for answers with more than five or six digits of accuracy. Much of the input data comes from physical measurements (6.34 pounds, 18.97 inches, etc.) that are only approximate anyway, so the resulting answers (with some exceptions) must also be considered approximate.

However, in an accounting application, $713,403.14 is exactly that—$713,403.14. If you add up your sales by area, by salesman, by item, by customer, etc., the grand total for each had better be the same, right down to the last penny.

For this reason, commercial programmers must be familiar with the ways the 1130 does arithmetic, and aware of their advantages and disadvantages.

For purposes of discussion, three are four ways to do arithmetic on the 1130 system:

- Integer mode
- Real mode, floating point
- Real mode, fixed point
- Decimal mode
Integer Mode

An integer is defined as a whole number, a number with no fractions. Using 1130 FORTRAN, integers are limited to a magnitude of +32767 to -32768. This range is due to the fact that an integer must fit in one 16-bit word. 32767 is the largest positive number that can fit in one word (0111111111111111, where the first bit represents the sign); -32768 is the largest negative number.

Because of these two limitations (magnitude, and lack of fractions) you must be careful in your use of integer mode arithmetic. Integer mode is generally used for counters and indicators. However, if you desire to keep track of the position of the decimal point yourself, you can use integer arithmetic to process data with implied decimal points.

For example, if you know that pay rates at your company range from $1.25 to $6.50 per hour, you could represent these rates as integers ranging from 125 to 650 cents per hour. If rates ranged from $1.250 to $6.500 per hour, with some rates involving fractions of cents (say $3.375 per hour), they could be represented as integers from 1250 to 6500 mills per hour.

Since mixed mode arithmetic is permitted in 1130 FORTRAN, there is no problem involved in multiplying the integer IRATE by the real HOURS:

\[ \text{PAY} = \text{HOURS} \times \text{IRATE} \]

If IRATE is 3125 ($3.125 per hour) and HOURS is 33.5, PAY will be 104687.5. After the multiplication you must be careful to reposition the decimal point in the proper place ($104.6875) and round off ($104.69) before printing the result or accumulating totals.
Real Mode

General

A real number may be defined as a number with a decimal point; fractions are allowed. If you use 1130 FORTRAN for real arithmetic, the arithmetic subroutines will keep track of the decimal point for you, and the output subroutines will place it in the proper place in the output results.

On the 1130, a real number may be thought of as having four components:
1. The whole portion
2. The fractional portion
3. A pointer indicating the location of the decimal point
4. A positive or negative sign

For example, the number 267.4 has:
1. A whole portion, 267
2. A fractional portion, .4
3. A pointer indicating that the decimal point is between the 7 and the 4
4. A positive sign

Since the 1130 is a binary computer, these four components are represented in binary form as follows:
- The 267 as 100001011
- The .4 as .011001100110----
- A pointer of 9 showing that the binary point is between the ninth and tenth bits
- The sign is positive (a 0 bit)

Rearranging and simplifying somewhat, this can be written as (9, +, 100001011, .011001100110----)

The first value, the 9, is in decimal, the number of bits in the whole portion; the second item is the sign; the third value is the whole portion itself; the last value is the fraction.

The number of bits available for the whole and fraction combined depends on the precision option selected:
- Standard precision allots 23 bits for these two items.
- Extended precision allots 31 bits for them.

The whole portion of the number, since it is more significant, gets first choice of the available bits. In this case, the whole portion (267) requires 9 bits, leaving either 14 or 22 bits for the fraction, depending on the precision chosen.

This can cause inaccuracies, since most fractions cannot be represented exactly in 14 or 22 bits, or in any number of bits, for that matter. To see why, let us see how .4 in the above example is represented in binary notation. You are probably familiar with the binary system for whole numbers (1, 2, 4, 8, 16, 32, etc., or \(2^0, 2^1, 2^2, 2^3, 2^4, 2^5\), etc., respectively). In the case of fractions, the values proceed from the decimal (or binary) point to the right as \(1/2, 1/4, 1/8, 1/16, 1/32\), etc., or \(2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, 2^{-5}\), etc., respectively.

For example, .625 is

\[.101000000\]

or 
\[1/2 \text{ plus } 1/8\]

or 
\[.5000 \text{ plus } .125\]

It can be represented exactly in only three bits; however, this is unusual.

The example, .4, appears to be a rather simple number, and you might think that it also can be represented exactly as a binary fraction. The table below shows that this is not true:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Position</th>
<th>Value</th>
<th>Used = 1</th>
<th>Not Used = 0</th>
<th>Subtotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.5</td>
<td>0</td>
<td>0</td>
<td>.0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.25</td>
<td>1</td>
<td>.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.125</td>
<td>1</td>
<td>.375</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.0625</td>
<td>0</td>
<td>.375</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.03125</td>
<td>0</td>
<td>.375</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>.015625</td>
<td>1</td>
<td>.390625</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>.0078125</td>
<td>1</td>
<td>.3984375</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>.00390625</td>
<td>0</td>
<td>.3984375</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>.001953125</td>
<td>0</td>
<td>.3984375</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>.0009765625</td>
<td>1</td>
<td>.3994140625</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>.00048828125</td>
<td>1</td>
<td>.39990234375</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>.000244140625</td>
<td>0</td>
<td>.39990234375</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

You see that the binary representation can come close to .4 but never hit it. With 12 bits (.011001100110) the decimal value is .39990234375; with 16 bits, .399993896484375; with 20 bits, .399999618530234375; etc.

The fraction chosen, .4, is not an unusual number; it is typical of most fractions.

Unlike fractions, whole numbers can be represented exactly in binary form. However, you do reach a limit, depending on the number of bits available. In standard precision, if you use all 23 bits for the whole portion, you can attain a magnitude of 8,388,607. With extended precision, the 31 available bits yield 2,147,483,647.
Real—Floating Point

The term "floating point" implies that the decimal point is permitted to "float" among the digits in a real number. In other words, the 1130 arithmetic subroutines will keep track of the location of the decimal point and move it about to maintain the validity of the number. If you multiply $1.78 per hour by 32.20 hours, the answer becomes $57.3160. The decimal point "floats", thus remaining correctly positioned at all times.

As you saw before, though, the result may not be exact, since .316 probably cannot be represented exactly as a binary number. In fact, the 1.78 and the 32.20 were both probably inexact, too.

If you are doing an engineering or other non-commercial job, the answer is probably good enough; it matters little whether the result is 57.316000 or 57.31603 or 57.315999. If your application is commercially oriented, however, close is not good enough, since you are probably dealing with cash. Because accounting balances are so important, answers must be exact, down to the last unit (penny, box). It is not that people will worry about the penny itself, but that unbalanced totals traditionally indicate an error. If the face value of 600 payroll checks totals $12345.67, while the system's grand total is $12345.68, something may be seriously wrong somewhere. The fact that the net error is only one cent is immaterial; there may be 300 people overpaid by one cent, and 299 underpaid by one cent.

Real—Fixed Point

To eliminate the inaccuracies described above, you can use real arithmetic in a "fixed point" mode. "Fixed point" means that the decimal point is kept fixed to the right of the least significant digit, eliminating fractions altogether.

Earlier, you saw that 1.78 times 32.20 gave 57.3160, an answer that probably was inexact. If, however, you had used real, fixed point arithmetic, you would have multiplied 1A78. by 3A220., and obtained 57A3160., exactly. All three numbers, since they involve no fractions, will be exact, not just close. Note that the A is used to locate the implied decimal point.

This puts a slight burden on your programmer: instead of letting the subroutines keep track of the decimal point for him, he must now do it himself.

Using the values mentioned above, 1.78 times 32.20 is 57.3160 (dollars), while 1A78. times 32A20. is 57A3160. (ten thousandths of dollars). In the latter case, you must remember that the true decimal point is four places to the left of the one supplied by the system.
Rounding

Suppose you have just calculated an employee's gross pay as 107,5673. (understood to be $107.5673) and wish to apply a deduction of $19.733 (represented as 19,733.). Notice that the decimal points are not "lined up"; the units are not the same—the gross is in hundredths of cents, and the deductions are in tenths of cents.

How do you perform this subtraction?

a. 107,5673. - (19,733. x 10.)
b. (107,5673. /10.) - 19,733.
c. (107,5673./10000.) - (19,733./1000.)

None of these is correct. Before subtracting, you must round these two quantities—commonly to the nearest cent.

In the case of the 107,5673. gross, you must add 1/2 cent or 50. hundredths, obtaining 107,5723., then divide by 100, to get 107,57.23. Now, since the .23 is both inexact and meaningless, it must be eliminated. The WHOLE function supplied with the Commercial Subroutine Package converts the fractional part of the number to zeros.

All three functions—round, shift and clear fractions—can be done in one statement. The statement

\[
GROSS = WHOLE ((GROSS + 50. / ABS(GROSS)* 0.5) / 10000 + .5)
\]

rounds off GROSS, shifts it two places to the right, and clears everything remaining to the right of the decimal point to zeros. Note that multiplication rather than division was used (see Section 70, 50.00).

In the case of the deduction, you would say

\[
DEDUC = WHOLE ((DEDUC + 5. / ABS(DEDUC)* 0.5) / 10000 + .5)
\]

Now both values have been rounded and are in whole cents, with all extraneous fractions cleared. Note what would have happened if the fractions had not been cleared:

\[
\begin{align*}
10757.23 \\
1973.80 \\
8783.43
\end{align*}
\]

The correct answer is:

\[
\begin{align*}
10757.00000 \\
1973.00000 \\
8784.00000
\end{align*}
\]

You may wish to code several arithmetic statement functions, each one shifting a predetermined number of places to the right:

\[
\begin{align*}
\text{RND1}(X) &= WHOLE((X + X /\text{ABS}(X)*5.0)/10.0 + 0.5) \\
\text{RND2}(X) &= WHOLE((X + X /\text{ABS}(X)*50.0)/100.0 + 0.5) \\
\text{RND3}(X) &= WHOLE((X + X /\text{ABS}(X)*500.0)/1000.0 + 0.5)
\end{align*}
\]

etc.

where the fourth character of the FUNCTION name indicates the number of places to be shifted.

Accuracy and Magnitude

Suppose you are using extended precision real numbers, where 31 bits are available for the whole number and fraction combined. How large a number can you have? 2,147,483,647? No, that is just the largest number that can fit in 31 bits. Values much larger are possible—for example, 1,000,000,000,000,666,777,888., which can easily be handled in the 1130.

The decimal point indicator can be as large as 64 in binary, or about 38 in decimal, meaning that extremely large real numbers can be represented on the 1130.

The drawback is their accuracy. Especially in commercial applications, numbers must be precise. The number 1,000,000,000,000,666,777,888. can be read into the 1130, but it will be accurate only to nine or ten decimal digits. In other words, the nine most significant digits will be retained, but the remaining digits will be lost. The decimal point indicator will show the proper magnitude, but the number is not accurate.

If you want accurate results, you must not exceed the 31 bits (2,147,483,647.) or 23 bits (8,388,607.) available.

Furthermore, if you want accurate numbers, you must not allow any fractions to be generated.

Combining the above two warnings, then, means that you should limit real numerical values to the whole numbers between -2,147,483,648. and +2,147,483,647. Any number outside this range will probably not be exact; most fractions will probably be inexact.

If you work commercial problems in cents, you are limited to $21,474,836.47 (carried as a whole, fixed point real number). The limit is $2,147,483.647 if you wish to work in mills.

These limits are usually ample for jobs such as payroll, etc., but may be troublesome in accounting-type work, where year-to-date sales, total assets, etc., may exceed $21 million. If this is the case, the decimal arithmetic subroutines of the 1130 Commercial Subroutine Package may be used.
Output of Large Real Numbers

A second precaution must be taken with very large numbers, and it falls in the area of output. Because most fractions are inexactly represented and will always be less than the true decimal value, the FORTRAN output routines (including the TRACE) always add a single bit in the low-order position of the number, attempting to compensate for this inaccuracy. For this reason, you rarely notice the inaccuracy.

For example, if you multiplied 0.35 by 100.0, you would expect to get 35.0, exactly. The binary result, however, converted to decimal, is

34.9999999999999999757138713...

(because the multiplier of 0.35 is an inexact fraction). That is not the result you see, though, since the FORTRAN output routine adds its one low-order bit, resulting in

35.000000029802322364091838...

Although that is no more exact than the previous value, it looks better; in fact, you would never notice the extra

0.000000298023223876953125

unless you output the number with a format like F40.20, which would be unusual.

If your number is large, however, this "little extra" can cause the output to be noticeably in error. For example, the whole number 2111111111., while represented exactly in core storage, will be output as 2111111112., an error of 1. unit.

This problem will occur with numbers in the range 1,073,741,824. to 2,147,483,647. If they are output with a standard FORTRAN F format. For this reason, you may wish to limit the magnitude of all numbers to 1,073,741,824., or, easier to remember, nine digits (999,999,999.)

This problem does not occur if the value is printed as alphabetic data, converted by the PUT subroutine of the Commercial Subroutine Package. This routine will not add the extra bit, and all whole numbers up to 2,147,483,647. will be output exactly.

Multiplication of Large Real Numbers

Because of the manner in which the 1130 performs multiplication, a product is accurate only to 30 bits. This means that a product exceeding 1,073,741,823. may not be exactly correct. In fact, there is a 75-25 chance that it will be correct, and a 25-75 chance that it will be off by 1. unit.

While this is quite satisfactory for technical work, it cannot be permitted in most commercial applications. For this reason, you should avoid multiplications that might yield such a large product.

Note that this limitation does not apply to addition and subtraction, where all 31 bits may be used - and the upper limit is 2,147,483,647.
Decimal Mode

Introduction

In addition to integer and real mode arithmetic, there is a third alternative: decimal arithmetic. This capability is furnished by a group of subroutines supplied with the Commercial Subroutine Package (1130-SE-25X). This mode of arithmetic permits variable precision.

Using the decimal arithmetic system, you select the number of digits to be used for each variable. If a grand total can attain a magnitude of 15,000,000,000.00, you can allocate 13 digits for it; if the number of employee dependents never exceeds 99, you may allocate only two digits for that value.

You are not limited by magnitudes of 32767, 2,147,483,647, etc. You decide how large a number can become and set aside enough digits for its storage.

General Principles

This arithmetic system operates on digits stored as integers, one digit per word. For example, the value 1968 would be stored in a four-position array

NYEAR as

NYEAR (1) = 1
NYEAR (2) = 9
NYEAR (3) = 6
NYEAR (4) = 8

or in a six-position array as

NYEAR (1) = 0
NYEAR (2) = 0
NYEAR (3) = 1
NYEAR (4) = 9
NYEAR (5) = 6
NYEAR (6) = 8

or in any size array you desire.

Negative values carry the minus sign with the low-order (or rightmost) digit. However, since the 1130 cannot represent -0, a special method has been devised to show negative numbers. If the number is negative, the low-order digit is carried as one less than its true value.

For example, -1968 is actually held in core storage as

NYEAR (1) = 1
NYEAR (2) = 9
NYEAR (3) = 6
NYEAR (4) = -9

For ease of reference, we will refer to this as 1968, where the minus sign is written over the low-order digit.

You need not worry about this unless you are defining negative constants, which should be unusual. If a negative number is read from a card, this conversion will be done for you, with the A1DEC subroutine.

The magnitude of each value will be shown by the number of digits: 001968 implies a six-digit or six-word value, 0000001968 implies a ten-word value.

Each decimal arithmetic value requires three identifying parameters:

• The NAME of the array in which it is stored.
• KSTRT, the position (or subscript) of the high-order (leftmost) digit in the array.
• KLAST, the position of the low-order digit in the array.

When referring to decimal arithmetic values, a shorthand abbreviation will be used, enclosing these three parameters in parentheses:

(NAME, KSTRT, KLAST)

For example, if you had a 20-word array called NUMBR:

| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| 0 0 0 1 3 6 4 3 0 0 7 7 8 3 0 0 8 1 3 |

then

(NUMBR, 1, 6) = 000136 or -136
(NUMBR, 1, 5) = 00013 or 13
(NUMBR, 7, 19) = 430077830081
(NUMBR, 15, 20) = 000813 or -813
The Decimal Arithmetic Subroutines

The IBM 1130 Commercial Subroutine Package furnishes subroutines to perform the following four arithmetic operations:

- **ADD** to add two decimal values
- **SUB** to subtract two decimal values
- **MPY** to multiply two decimal values
- **DIV** to divide two decimal values

All four have similar calling sequences, requiring three basic elements:

- The identification of the first variable
- The identification of the second variable
- An error code

Since the identification of each variable requires three parameters (e.g., NUMBR, 1, 6), each subroutine has a total of seven parameters.

If no error conditions occur, the subroutine leaves the error code, NER, set to whatever value it had when the subroutine was called. Note that the subroutines merely set the indicator NER. They do not pause, print a message, or take any other definite action. It is up to you to set NER before calling the subroutines, and to test it after each is complete.

**Addition.** The general form of the ADD subroutine is

\[
\text{CALL ADD (addend, augend, error code)}
\]

where the addend is added to the augend, and the result is left in the augend.

There are two possible error conditions. Both are illustrated in the accompanying examples (Figures 70.1 through 70.6).

**Subtraction.** The general form of the Subtract subroutine is

\[
\text{CALL SUB (subtrahend, minuend, error code)}
\]

where the subtrahend is subtracted from the minuend, and the result replaces the minuend.

There are two possible error conditions. Both are included in the accompanying examples (Figures 70.7 through 70.11).

**Multiplication.** Because of its nature, multiplication is somewhat more involved than addition and subtraction. For example, if you multiply two two-digit numbers, 95 and 86, your result is 8220, a four-digit number. If you multiply a three-digit number, 666, by a two-digit number, 55, your answer is 36630, a five-digit number. The result of a multiplication, the product, may have as many digits as the sum of the number of digits in the multiplier and the multiplicand. Therefore, you must provide that many digits for the result.

The MPY subroutine accomplishes this in a very straightforward manner. The multiplicand field, which will be the eventual location of the product, is extended to the left the same number of digits as are contained in the multiplier. For example, if you multiply a four-digit number by a two-digit number, the subroutine will extend the four-digit field to the left two places, to hold the six-digit product.

It does this regardless of what was in these positions previously. Obviously, you must consider this fact when laying out your data areas in core storage.

Figures 70.12 and 70.13 present several examples of the use of the MPY subroutine.

**Division.** The divide subroutine, DIV, has the calling sequence

\[
\text{CALL DIV (divisor, dividend, error code)}
\]

with the result placed in the dividend field.

Before covering the DIV subroutine, a quick review of division might be in order. If you divide 13 by 4, the result is 3 1/4, where

- 13 is the dividend
- 4 is the divisor
- 3 is the quotient
- 1 is the remainder

In other words:

\[
\frac{\text{dividend}}{\text{divisor}} = \text{quotient} + \frac{\text{remainder}}{\text{divisor}}
\]

This is the form of the result after use of DIV -- a quotient of 3 and a remainder of 1. Note that the result is not 3.25. For this reason special care must be taken when half-adjusting the result of a division. Also note in Figures 70.14 through 70.17 that the length of the remainder field will be the same as the length of the divisor.
1130 COMMERCIAL SUBROUTINES — DECIMAL ARITHMETIC WORKSHEET

DESCRIPTION

BEFORE

X = Extraneous Data

IWORK: Will remain unchanged

KWORK: Will contain result

CODING

CALL

AFTER

IWORK: Unchanged

KWORK: Result

COMMENTS
**1130 COMMERCIAL SUBROUTINES — DECIMAL ARITHMETIC WORKSHEET**

**DESCRIPTION**

**ADDITION OF TWO FIVE-DIGIT FIELDS**

<table>
<thead>
<tr>
<th>BEFORE</th>
<th>X = Extraneous Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWORK: Will remain unchanged</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

| KWORK: Will contain result |

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**CODING**

NER = \[ \boxed{0} \]

CALL ADD \[ (IWORK, 1, 5, KWORK, 4, 8, NER) \]

<table>
<thead>
<tr>
<th>AFTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWORK:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

| KWORK: Result |

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>9</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**COMMENTS**

NER IS STILL 0, SO THE ADDITION WAS CORRECT.

Figure 70.1.
### DESCRIPTION

**ADDITION OF TWO 4-DIGIT FIELDS, WHERE ONE IS NEGATIVE. SUM IS POSITIVE**

### BEFORE

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**X = Extraneous Data**

**IWORK:** Will remain unchanged

**KWORK:** Will contain result

**THE 6 IS ACTUALLY IN CORE STORAGE AS -7**

### CODING

- **NER = 0**
- **CALL ADD (IWORK, 5, 8, KWORK, 1, 4, NER)**

### AFTER

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IWORK:**

**KWORK:** Result

**NER = 0**

### COMMENTS

Figure 70, 2
1130 COMMERCIAL SUBROUTINES – DECIMAL ARITHMETIC WORKSHEET

DESCRIPTION
ADDITION OF TWO G-DIGIT FIELDS, WHERE ONE IS NEGATIVE. SUM IS NEGATIVE

BEFORE
X = Extraneous Data

IWORK: Will remain unchanged

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

KWORK: Will contain result

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE 5 IS IN CORE STORAGE AS -6

CODING

NER = 0

CALL ADD (IWORK, 1, G, KWORK, 1, G, NER)

AFTER

IWORK:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

KWORK: Result

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

3 IN CORE AS -6

COMMENTS

Figure 70.3.
**DESCRIPTION**

**ADDITION OF A 1-DIGIT FIELD TO A 5-DIGIT FIELD, BOTH POSITIVE**

**BEFORE**

\[ X = \text{Extraneous Data} \]

**IWORK:** Will remain unchanged

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**KWORK:** Will contain result

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**CODING**

<table>
<thead>
<tr>
<th><strong>NER</strong> = 0</th>
</tr>
</thead>
</table>

**CALL** \( \text{ADD} (\text{IWORK}, 5, 5, \text{KWORK}, 1, 5, \text{NER}) \)

**AFTER**

**IWORK:**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**KWORK:** Result

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>8</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**COMMENTS**

Figure 70, 4.
1130 COMMERCIAL SUBROUTINES – DECIMAL ARITHMETIC WORKSHEET

DESCRIPTION - ADDITION OF TWO 2-DIGIT FIELDS
- SUM DOES NOT FIT IN ALLOTTED AREA

BEFORE  X = Extraneous Data

IWORK: Will remain unchanged

KWORK: Will contain result

CODING

NER = 0

CALL ADD (IWORK, 4, 5, KWORK, 2, 3, NER)

AFTER

IWORK:

KWORK: Result

COMMENTS - THE SUM AREA IS FILLED WITH 9's
- NER IS SET TO THE VALUE OF THE G-th PARAMETER

Figure 70.5.
### 1130 COMMERCIAL SUBROUTINES — DECIMAL ARITHMETIC WORKSHEET

**DESCRIPTION**
Addition of two fields, where the **addend is longer than the augend**

**BEFORE**
X = Extraneous Data

<table>
<thead>
<tr>
<th>IWORK: Will remain unchanged</th>
<th>KWORK: Will contain result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 6 8 0 X X X X X</td>
<td>0 0 0 8 X X X X X X X X</td>
</tr>
</tbody>
</table>

**CODING**

**NER**

\[
\text{CALL ADD (IWORK, 1, 5, KWORK, 1, 4, NER)}
\]

**AFTER**
NER = 4

<table>
<thead>
<tr>
<th>IWORK:</th>
<th>KWORK: Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 6 8 0 X X X X X</td>
<td>0 0 0 8 X X X X X X X</td>
</tr>
</tbody>
</table>

**COMMENTS**

Note - Even though the result, 688, would fit in 4 digits, the ADD subroutine will not add, since this is a potentially dangerous situation. NER is set to 4.
### Description

**Subtract one 5-digit field from another 5-digit field**

### Before

<table>
<thead>
<tr>
<th>IWORK</th>
<th>KWORK</th>
</tr>
</thead>
<tbody>
<tr>
<td>X X X X 0 0 0 1 8 X X</td>
<td>X X X X X X X X 0 0 0 1 4</td>
</tr>
</tbody>
</table>

- **X** = Extraneous Data
- **IWORK**: Will remain unchanged
- **KWORK**: Will contain result

### Coding

- **NER** = 0
- **CALL SUB** (IWORK, 5, 9, KWORK, 9, 13, NER)

### After

<table>
<thead>
<tr>
<th>IWORK</th>
<th>KWORK</th>
</tr>
</thead>
<tbody>
<tr>
<td>X X X X 0 0 0 1 8 X X</td>
<td>X X X X X X X X 0 0 0 0 4</td>
</tr>
</tbody>
</table>

- **NER** = 0

### Comments

Figure 70.7.
1130 COMMERCIAL SUBROUTINES — DECIMAL ARITHMETIC WORKSHEET

DESCRIPTION
SUBTRACT ONE 10-DIGIT FIELD FROM A 12-DIGIT FIELD. RESULT IS NEGATIVE

BEFORE
X = Extraneous Data

<table>
<thead>
<tr>
<th>IWORKE</th>
<th>Will remain unchanged</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11</td>
<td></td>
</tr>
<tr>
<td>0000000000 X</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KWORK</th>
<th>Will contain result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13</td>
<td></td>
</tr>
<tr>
<td>X0000055555555</td>
<td></td>
</tr>
</tbody>
</table>

CODING

NER = 0
CALL SUB(IWORK, 1, 10, KWORK, 2, 13, NER)

AFTER

<table>
<thead>
<tr>
<th>IWORKE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11</td>
<td></td>
</tr>
<tr>
<td>0000000000 X</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KWORK</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13</td>
<td></td>
</tr>
<tr>
<td>X000000111111</td>
<td></td>
</tr>
</tbody>
</table>

COMMENTS

1 IS IN CORE AS -2

Figure 70.8.
1130 COMMERCIAL SUBROUTINES - DECIMAL ARITHMETIC WORKSHEET

DESCRIPTION
SUBTRACT A 3-DIGIT FIELD FROM A 2-DIGIT FIELD.

BEFORE
X = Extraneous Data

IWORK: Will remain unchanged

KWORK: Will contain result

CODING
SUB
CALL
NER = 0

AFTER
NER = 7

COMMENTS
NOTE: -NER SET TO 7
- SUBTRACTION NOT CARRIED OUT BECAUSE OF POTENTIAL ERROR CONDITION

Figure 70.9.
## 1130 COMMERCIAL SUBROUTINES — DECIMAL ARITHMETIC WORKSHEET

### DESCRIPTION

**SUBTRACT A NEGATIVE 3-DIGIT FIELD FROM A POSITIVE 4-DIGIT FIELD**

### BEFORE

- **X** = Extraneous Data
- **IWORK**: Will remain unchanged
- **KWORK**: Will contain result

### CODING

NER = \( \frac{0}{0} \)

**CALL SUB** (IWORK, 6, 8, KWORK, 5, 8, NER)

### AFTER

- **IWORK**: Result
- **KWORK**: Result

### COMMENTS

Figure 70, 10.
**DESCRIPTION**

SUBTRACT A NEGATIVE 2-DIGIT FIELD FROM A POSITIVE 2-DIGIT FIELD. RESULT TOO LARGE.

**BEFORE**

X = Extraneous Data

IWORK: Will remain unchanged

KWORK: Will contain result

**CODING**

NER = \( \text{ADDEND} \rightarrow \text{AUGEND, THEN SUM} \rightarrow \text{MINUEND, THEN SUM} \)\

CALL SUB (IWORK, 4, 5, KWORK, 3, 4, NER)

**AFTER**

NER = 4

**COMMENTS**

NOTE - RESULT FIELD FILLED WITH 9's - NER SET TO 4
1130 COMMERCIAL SUBROUTINES – DECIMAL ARITHMETIC WORKSHEET

DESCRIPTION

MULTIPLY TWO 4-DIGIT NUMBERS

1111 * 2222 = 02468642

BEFORE

X = Extraneous Data

IWORK: Will remain unchanged

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

KWORK: Will contain result

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

CODING

NER = 0

CALL MPY (IWORK, 1, 4, KWORK, 5, 8, NER)

AFTER

NER = 0

IWORK: Unchanged

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

KWORK: Result

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

COMMENTS

NOTE THAT THE PRODUCT AREA (KWORK) HAS BEEN EXTENDED 4 PLACES TO THE LEFT.
MULTIPLY TWO 4-DIGIT NUMBERS, SHOWING AN ERROR CONDITION

BEFORE

X = Extraneous Data

IWORK: Will remain unchanged

KWORK: Will contain result

CODING

CALL MPY (IWORK, 1, 4, KWORK, 3, 6, NER)

AFTER

NER = 0

IWORK: Unchanged

KWORK: Result

COMMENTS

NER IS SET TO 6 BECAUSE THERE WAS NOT ENOUGH ROOM TO EXTEND THE PRODUCT AREA 4 PLACES TO THE LEFT
**DESCRIPTION**

DIVIDE 013 by 04

**BEFORE**

X = Extraneous Data

<table>
<thead>
<tr>
<th>IWORK: Will remain unchanged</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
<tr>
<td>X X X X X 0 4 X X X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KWORK: Will contain result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13</td>
</tr>
<tr>
<td>X X 0 1 3 X X X X X X X</td>
</tr>
</tbody>
</table>

**CODING**

NER = 0

CALL DIV (IWORK, 6, 7, KWORK, 3, 5, NER)

**AFTER**

<table>
<thead>
<tr>
<th>IWORK: Unchanged</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
<tr>
<td>0 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KWORK: Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13</td>
</tr>
<tr>
<td>0 0 3 0 1</td>
</tr>
</tbody>
</table>

**COMMENTS**

"BECAUSE THE DIVISOR IS 2 DIGITS WIDE, THE KWORK FIELD HAS BEEN EXTENDED 2 POSITIONS TO THE LEFT, AND THE REMAINDER OCCUPIES THE RIGHTMOST 2 DIGITS."

Figure 70.14
### 1130 COMMERCIAL SUBROUTINES — DECIMAL ARITHMETIC WORKSHEET

#### DESCRIPTION

**DIVIDE 015 BY 008**

**BEFORE**  
X = Extraneous Data

<table>
<thead>
<tr>
<th>IWORK: Will remain unchanged</th>
<th>KWORK: Will contain result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13</td>
</tr>
<tr>
<td>0 0 8 X X X X X X X X X X</td>
<td>X X X 0 1 5</td>
</tr>
</tbody>
</table>

**CODING**

```
CALL DIV (IWORK, 1, 3, KWORK, 4, 6, NER)
```

**AFTER**

<table>
<thead>
<tr>
<th>IWORK: Unchanged</th>
<th>KWORK: Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13</td>
</tr>
<tr>
<td>0 0 8 X X X X X X X X</td>
<td>0 0 1 0 0 7</td>
</tr>
</tbody>
</table>

**COMMENTS**

"BECAUSE THE DIVISOR IS 3 DIGITS WIDE, THE KWORK FIELD HAS BEEN EXTENDED 3 POSITIONS TO THE LEFT, AND THE REMAINDER OCCUPIES THE RIGHTMOST 3 DIGITS."

---

Figure 70, 15.
### 1130 COMMERCIAL SUBROUTINES — DECIMAL ARITHMETIC WORKSHEET

**DESCRIPTION**  
**D/V/DE A NEGA7/VE NUMBER BY A**  
**POSITIVE NUMBER**  
**-5/2**  
**=**  
**-2**  
**-1**  
**/2**  
**OR**  
**-3**  
**72**

**BEFORE**  
**X = Extraneous Data**

| IWORK: Will remain unchanged |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 0 | 2 | X | X | X | X | X | X | X | X | X |

| KWORk: Will contain result |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 0 | 0 | 5 | 

**CODING**

<table>
<thead>
<tr>
<th>NER = 0</th>
</tr>
</thead>
</table>

**CALL D/IV (IWORK, 1, 2, KWORk, 3, 5, NER)**

**AFTER**

| IWORK: Unchanged |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 0 | 2 | X | X | X | X | X | X | X | X | X |

| KWORk: Result |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 0 | 0 | 5 | 0 | 1 |

**COMMENTS**  
**- THE SIGN IS CARRIED WITH THE QUOT.**

---

Figure 70.16.
### Description

**Division by Zero is Invalid**

### Before

X = Extraneous Data

- **IWORK**: Will remain unchanged

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWORK</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Coding

\[ \text{NER} = 0 \]

**CALL DIV** (IWORK, 1, 3, KWORK, 4, 6, NER)

### After

- **IWORK**: Unchanged

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWORK</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- **KWORK**: Result

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>KWORK</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>7</td>
<td>0</td>
<td>8</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Comments

- No division is performed
- NER is set to 6
Constants

There are four ways in which you may create constants such as 1968, 40, 6600, etc. To illustrate, suppose you wish to create the constant 660000 (the Social Security deduction base, in cents) to be stored in an array named ISSD, DIMENSIONed as ISSD (6). The four options are:

1. Use FORTRAN equalities.
   
   \[
   \begin{align*}
   \text{ISSD (1)} &= 6 \\
   \text{ISSD (2)} &= 6 \\
   \text{ISSD (3)} &= 0 \\
   \text{ISSD (4)} &= 0 \\
   \text{ISSD (5)} &= 0 \\
   \text{ISSD (6)} &= 0 
   \end{align*}
   \]

2. Use the DATA statement.
   
   DATA ISSD/6, 6, 0, 0, 0, 0/
   or
   DATA ISSD/2*6, 4*0/

3. Use the FILL subroutine.
   
   CALL FILL (ISSD, 1, 2, 6)
   CALL FILL (ISSD, 3, 6, 0)

4. Read it from a card, tape, keyboard, or disk.

Option 2 is preferred, since it consumes less core storage than the other three methods.

Negative constants are handled in much the same way. Because of their special representation, however, it would be wise to make the constants positive and change the arithmetic. For example, rather than set up \(-1\) and add it to something, it would be easier to subtract \(+1\).

Testing and Modifying Signs

To facilitate testing and modifying the signs of decimal arithmetic fields, the subroutine NSIGN is available. It has four parameters:

- **NARRY**: The name of the array
- **NPOS**: The position in the array to be tested
- **NEWS**: "New sign", indicating what you want done to the previous sign:
  +1 Make it positive
  0 Reverse it
  -1 Make it negative
- **NOLDS**: "Old sign", returned to you, indicating what the previous sign was:
  +1 It was positive
  -1 It was negative

You, the programmer, send the subroutine the first three parameters; it returns the last. To illustrate, suppose you wish to test the sign of the 18th position in the K array:

- **Case 1**: It Is Now Positive:
  
  NOLDS is returned as +1
  K(18) is made + if you said
  CALL NSIGN (K, 18, +1, NOLDS)
  K(18) is changed to - if you said
  CALL NSIGN (K, 18, 0, NOLDS)
  K(18) is made - if you said
  CALL NSIGN (K, 18, -1, NOLDS)
  K(18) remains + if you said
  CALL NSIGN (K, 18, NOLDS, NOLDS)

- **Case 2**: It Is Now Negative:
  
  NOLDS is returned as -1 and
  K(18) is made + if you said
  CALL NSIGN (K, 18, +1, NOLDS)
  K(18) is changed to + if you said
  CALL NSIGN (K, 18, 0, NOLDS)
  K(18) is made - if you said
  CALL NSIGN (K, 18, -1, NOLDS)
  K(18) remains - if you said
  CALL NSIGN (K, 18, NOLDS, NOLDS)
Moving Signs

The NSIGN routine may also be used to move signs. The two statements

CALL NSIGN (NARRY, I, NOLD, NOLD)
CALL NSIGN (KARRY, J, NOLD, JUNK)

will make KARRY (J) have the same as NARRY (I).

Comparing Decimal Fields

The FUNCTION ICOMP is used to compare two variable length decimal fields. In practice, it is typically used within the parentheses of an IF statement;

IF (ICOMP (IWORK, 1, 5, KWORK, 6, 10)) 1, 2, 3

This statement will compare (IWORK, 1, 5) with (KWORK, 6, 10), and branch to
Statement 1 if the first is less than the second.
Statement 2 if they are equal.
Statement 3 if the first is greater than the second.

As was true with the ADD and SUB subroutines, the first field must not be longer than the second. Since no error code is returned from this subprogram, there is no way to tell that such an error has occurred, and the results will therefore be meaningless.
Summary

If exact results are desired, you must take certain precautions regarding arithmetic calculations.
1. Use one of the following:
   - Integer arithmetic
   - Decimal arithmetic
   - Real, fixed-point arithmetic, with no fractions
2. If fractions are allowed to occur (floating-point real arithmetic), your results are likely to show inaccuracies. These inaccuracies will be slight, but enough to cause significant problems.
3. If no number will ever exceed 8,388,607., you may use standard precision, real, fixed-point arithmetic.
4. If no addition will ever exceed 2,147,483,647., you may use extended precision, real, fixed-point arithmetic.
5. If the result of a multiplication will exceed 1,073,741,823., you should consider using decimal arithmetic, since real extended precision arithmetic will be inaccurate above this limit.
6. If the result of an addition or subtraction falls in the range 1,073,741,824. to 2,147,483,647., you should not attempt to output it with the standard FORTRAN F Format; use the PUT subroutine instead.
7. If any number will exceed 2,147,483,647. (now or in the foreseeable future), use decimal arithmetic rather than real arithmetic.
OVERLAPPED INPUT/OUTPUT

Introduction

As a machine, the IBM 1130 Computing System is capable of performing many tasks simultaneously. For example, it can print, type, read a card, and compute, all at the same time. This can be done through its "cycle-stealing" I/O channels and the priority interrupt system. Each I/O device may, through an interrupt, signal the CPU that it requires service, and steal a cycle (3.6 or 2.2 microseconds) from some other process to do what it needs done. This process is commonly referred to as "overlapping".

For example, in the case of the disk, one data word travels past the read/write heads every 27.8 microseconds. However, it only takes one cycle (3.2 or 2.2 microseconds) to transfer that word from core storage to the disk (if it is being written) or from the disk to core storage (if it is being read). This means that only a little more than 10% of the CPU time is required to read and write on the disk; the remainder is available for other use.

Although most of the 1130's I/O devices can be overlapped, standard 1130 FORTRAN permits only two of them to operate in this fashion: the disk and the 1403 Printer. There are several good reasons for this limitation. For example, suppose you wrote a program to read two numbers from a card, add them together, and print the result. With full overlap, the addition could conceivably be under way before the two numbers had even been placed in core. Obviously, this would not be satisfactory.

To take full advantage of the potential of the machine, in FORTRAN, it would be necessary to develop a special FORTRAN, which would then violate the USASI standards set up for that programming language. Avoiding this, IBM has developed the Commercial Subroutine Package (CSP) -- a set of subroutines operating within the FORTRAN system, rather than as part of the FORTRAN language itself.
The Commercial Subroutine Package Overlapped I/O Subroutines

CSP subroutines may be divided into three groups:
   The I/O subroutines themselves
   Several I/O utility subroutines
   Those character handling routines necessary for proper use of the I/O routines
This section discusses the former two groups; the latter is covered later in this section under "Character Handling Techniques".

General

All of the overlapped I/O subroutines operate on data in Al format -- one alphabetic character per word, left-justified. If you wish to read 80 card columns, you must set up an array 80 positions long to receive the data, and convert the Al data to whatever format you require for later processing. There are no FORMAT statements; you must handle all conversions (see "Character Handling Techniques").

Unlike standard FORTRAN, the overlapped I/O subroutines are oriented toward a sign punch over the low-order digit of a field. For example, a negative number or credit of -$6.50 would be punched with an 11-punch over the zero, rather than in a separate column, as would be done if FORTRAN FORMAT were used.

In general, for your non-disk I/O, you must choose either one system or the other: FORTRAN FORMAT or overlapped I/O. They may not be mixed within the same program.

For further detail on these subroutines, see the SRL manual H20-0241.

READ a Card, 1442-6 or 7

The subroutine READ will read a card from the 1442 Model 6 or 7, overlapping reading with the conversion from card code to Al format. The CPU will not proceed any further until the last desired card column has been read and converted. Therefore you need not be concerned that processing will be started before the desired values have reached core storage.

A typical call to this routine would be

\[
\text{NER} = -1 \\
\text{CALL READ (INOUT,1,80,NER)}
\]

which would read and convert 80 columns, and place the result in the array INOUT. It should be followed by a

\[
\text{IF (NER) xxx, xxx, xxx}
\]

If NER is still -1, everything is normal; if it is zero, the card just read was the last card in the hopper; if it is +1, there was a read or feed check (1442 malfunction).

It is equivalent to

\[
\text{DIMENSION INOUT(80)} \\
\text{77 FORMAT (80A1)} \\
\text{READ (2,77) INOUT}
\]
PUNCH a Card, 1442-6 or 7

The subroutine PUNCH will punch a card on the 1442 Model 6 or 7. Nothing will be overlapped with this activity. A typical use is

NER = -1
CALL PUNCH (INOUT, 1, 20, NER)

which will punch the first 20 words of the INOUT array into the first 20 columns of a card.

It is equivalent to

DIMENSION INOUT (80)
77 FORMAT (80A1)
WRITE (2, 77) (INOUT(K), K=1, 20)

The use of the error parameter, NER, is identical to the READ subroutine.

Select STACKer, 1442-6 or 7

Subroutine STACK permits the FORTRAN programmer to direct a card into the alternate stacker on the 1442 Model 6 or 7. After the statement

CALL STACK

the last card read (and only the last card) will be selected into the alternate stacker.

The placement of the CALL STACK statement is important:
- If the program reads and punches into the same card, the statement may be placed between the READ and WRITE, or after the WRITE.
- If the program reads (but doesn't punch), the CALL STACK goes after the READ statement that read the card to be stacked.
- If the program only punches (and does not read), the CALL STACK should be placed after the WRITE statement that punches the card to be stacked.
PRINT on 1132

Subroutine PRINT enables you to write on the 1132 Printer, overlapping printing with other processing. A typical use of this routine is

\[
\text{NER} = 1 \\
\text{CALL PRINT (INOUT, 1, 120, NER)}
\]

This will initiate the printing of the 120-word array INOUT on the 1132, then continue processing. Because of its overlapped capability, it can drive the 1132 Printer substantially faster than the equivalent FORTRAN/FORMAT statements:

\[
\text{DIMENSION INOUT (120)} \\
88 \text{ FORMAT (120A1)} \\
\text{WRITE (3, 88) INOUT}
\]

Like READ and PUNCH, it should be followed with a test of NER:
- If it is still 1, nothing unusual happened.
- If it is 3, the line being printed matches with a channel 9 punch on the carriage control tape.
- If it is 4, the line being printed matches with a channel 12 punch in the carriage control tape.

Note that the first position is not used to control the printer carriage, as it is with standard FORTRAN. The SKIP routine must be used if you wish to skip to channel 1, double-space, etc.

SKIP on 1132

Subroutine SKIP permits full use of the carriage control tape mechanism on the 1132. Skipping is significantly faster than printing blank lines and should be used wherever possible. A typical use of this routine is

\[
\text{CALL SKIP (KODE)}
\]

where the allowable values of KODE, and their meaning, are as shown in Figure 70.18.

<table>
<thead>
<tr>
<th>Value</th>
<th>Action Taken by the 1132</th>
</tr>
</thead>
<tbody>
<tr>
<td>12544</td>
<td>Immediate skip to channel 1</td>
</tr>
<tr>
<td>12800</td>
<td>Immediate skip to channel 2</td>
</tr>
<tr>
<td>13056</td>
<td>Immediate skip to channel 3</td>
</tr>
<tr>
<td>13312</td>
<td>Immediate skip to channel 4</td>
</tr>
<tr>
<td>13568</td>
<td>Immediate skip to channel 5</td>
</tr>
<tr>
<td>13824</td>
<td>Immediate skip to channel 6</td>
</tr>
<tr>
<td>14592</td>
<td>Immediate skip to channel 9</td>
</tr>
<tr>
<td>15360</td>
<td>Immediate skip to channel 12</td>
</tr>
<tr>
<td>15616</td>
<td>Immediate space of 1 space</td>
</tr>
<tr>
<td>15872</td>
<td>Immediate space of 2 spaces</td>
</tr>
<tr>
<td>16128</td>
<td>Immediate space of 3 spaces</td>
</tr>
<tr>
<td>0</td>
<td>Suppress space after printing</td>
</tr>
</tbody>
</table>

Figure 70.18. SKIP codes for 1132 Printer
Type on Console Printer

Subroutine TYPER will initiate typing on the console printer, and then continue processing. Actual printing time will be overlapped with other processing (printing on the 1132, reading cards, computing, etc.). A typical call is

\[ \text{CALL TYPER (INOUT, 1,50)} \]

which will type the first 50 values of the INOUT array. There is no error parameter connected with this routine.

In addition to printing, this subroutine also permits several typewriter control functions. If the values listed below are inserted in the INOUT array, the corresponding action will be performed at that point:

<table>
<thead>
<tr>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1344</td>
<td>Tabulate</td>
</tr>
<tr>
<td>5184</td>
<td>Shift to black</td>
</tr>
<tr>
<td>13632</td>
<td>Shift to red</td>
</tr>
<tr>
<td>5696</td>
<td>Backspace</td>
</tr>
<tr>
<td>5440</td>
<td>Carrier return</td>
</tr>
<tr>
<td>9536</td>
<td>Line feed</td>
</tr>
</tbody>
</table>

Because TYPER does not start each line with an automatic carrier return, you may want to place a 5440 in position 1 of the output array.

Accept Data from Console Keyboard

Subroutine KEYBD will read characters entered from the console keyboard. Only 60 characters at a time may be read with this routine. This activity is not overlapped with any other function. An example of the use of this subroutine is

\[ \text{CALL KEYBD (INOUT, 1,30)} \]

which will read 30 characters from the keyboard. This is no error parameter.
A Precaution -- IOND

Because of the properties of the overlapped I/O subroutines, all I/O activity must be completed before you cause the 1130 to PAUSE or STOP. The subroutine IOND will do this for you by testing the status of the interrupts and looping until none are pending.

IOND is required only when Version 1 of the Monitor is used; it should not be used if Version 2 of the Monitor is in use.

The call to IOND should always be placed immediately before each PAUSE or STOP statement:

```
CALL IOND
PAUSE 1234
```

or

```
CALL IOND
STOP 5678
```
Using The Overlapped I/O System

General

If you are to gain the full potential of the overlapped I/O routines, you should know several basic principles of this system:

- You must decide whether your non-disk I/O will be done by FORTRAN FORMAT READs and WRITEs or by the overlapped I/O subroutines. A program cannot use both. Note that the disk I/O is completely independent of the overlapped I/O system and does not enter into this discussion.

- Certain devices are not overlapped by these routines, making the placement of the subroutines CALLs quite important.

Overlapping and Your Program

As far as your program is concerned, only two I/O devices are really overlapped: the 1132 Printer and the console printer. The other devices are either not overlapped at all or overlapped with various housekeeping chores (for example, code conversion) rather than with your program. In other words:

These subroutines initiate an action, then continue processing:

<table>
<thead>
<tr>
<th>PRINT</th>
<th>SKIP</th>
<th>TYPER</th>
</tr>
</thead>
</table>

These subroutines start an action and finish it before they continue processing:

| READ | PUNCH | KEYBD |

Thus the sequence in which you use these routines becomes important. For example, suppose you have a program that develops some result, then must print a line on the 1132 and punch a card. How should this be done?

Alternative A

Develop results
CALL PRINT ( )
CALL PUNCH ( )

With alternative A, PRINTing is initiated, then PUNCHing, and the two I/O functions are overlapped. Alternative B, on the other hand, does not overlap these two functions, since the 1130 will wait until PUNCHing is completed before starting PRINTing. Alternative B does, however, overlap whatever follows the PRINT statement.

To gain maximum overlap, then, the three truly overlapped routines (PRINT, SKIP, and TYPER) should be placed as early in the processing cycle as possible. Figure 70.19 gives some examples of good and bad usage of these routines.

<table>
<thead>
<tr>
<th>Example</th>
<th>Bad Practice</th>
<th>Good Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>processing</td>
<td>processing</td>
</tr>
<tr>
<td></td>
<td>processing</td>
<td>processing</td>
</tr>
<tr>
<td></td>
<td>CALL PRINT</td>
<td>CALL SKIP</td>
</tr>
<tr>
<td></td>
<td>CALL PRINT</td>
<td>CALL PRINT</td>
</tr>
<tr>
<td>2</td>
<td>processing</td>
<td>processing</td>
</tr>
<tr>
<td></td>
<td>processing</td>
<td>processing</td>
</tr>
<tr>
<td></td>
<td>CALL PRINT</td>
<td>CALL PRINT</td>
</tr>
<tr>
<td></td>
<td>CALL PRINT</td>
<td>CALL PRINT</td>
</tr>
<tr>
<td>3</td>
<td>CALL PUNCH</td>
<td>CALL PRINT</td>
</tr>
<tr>
<td></td>
<td>CALL PRINT</td>
<td>CALL PRINT</td>
</tr>
<tr>
<td>4</td>
<td>WRITE disk</td>
<td>CALL PRINT</td>
</tr>
<tr>
<td></td>
<td>CALL PRINT</td>
<td>WRITE disk</td>
</tr>
</tbody>
</table>

Figure 70.19.
FORTRAN TRACE Not Permitted with Overlapped I/O Routines

If you use the overlapped I/O routines, you must not include any of the non-disk I/O devices on the *IOCS control record; only *IOCS (DISK) is permitted. This means that you may not take advantage of the standard FORTRAN TRACE facility, but must instead program your own trace. If this is done while the program is being developed, it presents little difficulty.

Several methods may be used -- for example:

- A series of numbered pauses, to display your progress through the program.
- A set of extra PRINT or TYPER statements, to function much the same as the standard TRACE. It might be useful to code a subroutine called TRACE, which would do this after testing Data Switch 15.

Alphabetic Headings with the Overlapped I/O System

Since you may not use FORMAT statements in conjunction with the overlapped I/O routines, you must enter alphabetic headings and other constants in some other manner. Several methods are available.

1. Use the DATA statement. This allows alphabetic constants to be entered, in the proper format, at the start of the program.

2. Read the alphabetic data from the card deck. You may lay out all the alphabetic data required (headings, error messages, etc.) so as to fit in one large array, then read that array from a deck of cards each time the program is executed. Because it is done only once, those program steps could be made into a LINK, in which case it could use FORTRAN I/O, regardless of which system the main program used.

3. Same as 2, except that the read-in program is run only once and places the array of heading information on the disk. This data is then read from the disk each time the main program is executed. This is somewhat more foolproof, since you do not have to worry about assembling the card deck each time the main program is run.
THE INTERACTION OF ARITHMETIC AND I/O

You have seen that two options are available for I/O:

Standard FORTRAN FORMAT
Overlapped I/O subroutines

You have also seen that, for all practical purposes, two options are available for arithmetic:

FORTRAN real arithmetic
Decimal arithmetic, variable length.

While you may choose any desired combination, certain combinations appear easier to use than others. You can see from Figure 70.21 that some provision must in all cases be made for conversion from input code to some arithmetic code, then from some arithmetic code to output code. If you use standard FORTRAN exclusively, you specify, with the FORMAT statement, what conversions you want. If you use any of the other three combinations, you must specify the desired code conversion with the character handling routines supplied by the Commercial Subroutine Package: GET, PUT, EDIT, DECA1, A1DEC, PACK, and UNPAC. (These routines are covered in later sections of this Guide.)

Figure 70.22 summarizes the advantages and disadvantages of each alternative. You can see that the convenience items (ease of programming, readability, etc.) are gradually sacrificed in order to make gains in the area of capability and performance.

<table>
<thead>
<tr>
<th></th>
<th>Convenience Items</th>
<th>Capability and Performance Items</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Easily Programmed</td>
<td>Easy to Readable Program</td>
</tr>
<tr>
<td>Standard FORTRAN</td>
<td>easy</td>
<td>very good</td>
</tr>
<tr>
<td>Standard FORTRAN Extended with GET, PUT and EDIT</td>
<td>a little harder</td>
<td>very good</td>
</tr>
<tr>
<td>Standard FORTRAN Arith, with GET, PUT and EDIT, and overlapped I/O</td>
<td>a little harder</td>
<td>very good</td>
</tr>
<tr>
<td>FORTRAN I/O with GET, PUT, EDIT and Decimal Arith.</td>
<td>a little harder</td>
<td>good</td>
</tr>
<tr>
<td>Overlapped I/O with Decimal Arith.</td>
<td>a little harder</td>
<td>good</td>
</tr>
</tbody>
</table>

Figure 70.22.
CHARACTER HANDLING TECHNIQUES

General

A great deal of the programming effort in most commercial applications falls into the general classification of character handling -- code conversion, editing, moving data, testing zone punches, comparing alphabetic data, etc. This section covers many of these tasks in detail, showing how they may be accomplished with the Commercial Subroutines.
Code Conversion

As you saw earlier, code conversion is essential to any program, commercial or technical. If you use standard FORTRAN, you must specify the desired conversions with the FORMAT statement. If you are using FORTRAN augmented by the Commercial Subroutines, you can also use the GET, PUT and EDIT subroutines for special formatting. If you are using the overlapped I/O routines, you must specify all the code conversions with the Commercial Subroutines (except A1 format), since no FORMAT statements may be used.

Basically there are five internal codes with which you might be concerned:

- Integer
- Real
- Alphabetic -- one character per word (A1)
- Alphabetic -- two characters per word (A2)
- Decimal -- one digit per word

Very few programs can avoid converting from one code to the other. Figure 70.23 shows the tools at your disposal to effect all possible conversions. The common ones are handled by a single subroutine; those less often needed require a combination of two or three subroutines.

The A1 code is particularly important since all the overlapped I/O routines require data in that format. In addition, GET, PUT, and EDIT work with data in the A1 format.

The A2 code is used primarily when writing alphabetic data on the disk, since it holds twice as much data per word as A1 format.

Decimal code is encountered only if you are using the decimal arithmetic, variable length routines of CSP.

<table>
<thead>
<tr>
<th>FROM</th>
<th>Integer</th>
<th>Real</th>
<th>Alpha (A1)</th>
<th>Alpha (A2)</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>FLOAT</td>
<td>PUT (FLOAT)</td>
<td>FLOAT, then PACK</td>
<td>FLOAT, then A1DEC</td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td>IFIX</td>
<td>PUT</td>
<td>PUT, then PACK</td>
<td>PUT, then A1DEC</td>
<td></td>
</tr>
<tr>
<td>Alphabetic (A1)</td>
<td>IFIX (GET)</td>
<td>GET</td>
<td>PACK</td>
<td>A1DEC</td>
<td></td>
</tr>
<tr>
<td>Alphabetic (A2)</td>
<td>UNPAC, then IFIX</td>
<td>UNPAC, then GET</td>
<td>UNPAC</td>
<td>UNPAC, then A1DEC</td>
<td></td>
</tr>
<tr>
<td>Variable Length</td>
<td>DECA1, then GET, then IFIX</td>
<td>DECA1, then GET</td>
<td>DECA1</td>
<td>DECA1, then PACK</td>
<td></td>
</tr>
</tbody>
</table>

Integer to Real -- FLOAT

The FLOAT function, a FORTRAN standard, is used to convert an integer to a real number. A typical use of this function is

\[ X = \text{FLOAT}(K) \]

which will set the real variable \( X \) equal to the value of \( K \). The same conversion can also be accomplished by coding

\[ X = K \]

This also uses the FLOAT function, even though it does not appear.

Real to Integer -- IFIX

The IFIX function, also a FORTRAN standard, is used to convert a real number to an integer. A typical use is

\[ K = \text{IFIX}(X) \]

which will take the real variable \( X \), convert it to an integer, and store it as \( K \). If \( X \) is 6.0, then \( K = 6 \); if \( X \) is 87.9, then \( K = 87 \); and so on.

This can also be accomplished by coding \( K = X \); this too uses the IFIX function.
A1 to Real -- GET

IBM supplies the GET function as part of the 1130 Commercial Subroutine Package (CSP). The original A1 data may have resulted from a FORTRAN READ with an A1 FORMAT, or from use of one of the CSP Overlapped Input routines, which always results in A1 format.

If you have a five-place array, in A1 format

\[
K(1) = 1 \\
K(2) = 9 \\
K(3) = 8 \\
K(4) = 6 \\
K(5) = 8
\]

and you say

\[
X = \text{GET}(K, 1, 5, 1.0)
\]

then \( X = 19868 \).

The last parameter (1.0) is a shift factor, and will usually be 1.0 if you want accurate results. (If it had been .1, \( X \) would be 1986.8; however, since the fraction .8 is present, you could expect it to be inaccurate.) Subsection 70.10.20 explains why fractions should be avoided in commercial work.

Basically, the above use of GET can be thought of as equivalent to an F5.0 in a FORMAT statement. A shift factor of .1 would be an F5.1; a shift of .01 would be F5.2; a shift of .001 would be F5.3; etc.

A1 to Integer

As shown in Figure 70.23, this step requires use of both IFIX and GET, in the following manner:

\[
J = \text{IFIX}(\text{GET}(K, 10, 12, 1.0))
\]

where positions 10 through 12 of the K array are converted first to a real number, then to an integer called \( J \).
Real to A1 -- PUT

This step is quite commonly required -- if you are using the overlapped I/O routines, if you wish to do further editing, etc. It is accomplished with the PUT subroutine supplied with CSP.

Suppose you have just computed a gross pay figure, GROSS, which might have a typical value of 275869.0, understood to be mills. Again, note that you are working in whole numbers, so that no fraction problems are encountered. This value is to be rounded off and the result placed in the first ten positions of array KGROS for later editing and output. The statement

```
CALL PUT (KGROS, 1, 10, GROSS, 5.0, 1)
```

will take GROSS, add 5.0 to it, truncate the last 1 digit, and place it in A1 format in the KGROS array as 0000027587, with leading zeros and no decimal point.

The last two parameters, the adjust factor and the truncate factor, usually form a logical pair. Obviously, if you add 5.0 to half-adjust, you won't want to print the resulting digit. The table below shows the common pairs:

<table>
<thead>
<tr>
<th>5th parameter (half-adjust factor)</th>
<th>6th parameter (how many digits to truncate from right end)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.5</td>
<td>0</td>
</tr>
<tr>
<td>5.</td>
<td>1</td>
</tr>
<tr>
<td>50.</td>
<td>2</td>
</tr>
<tr>
<td>500.</td>
<td>3</td>
</tr>
<tr>
<td>etc.</td>
<td>etc.</td>
</tr>
</tbody>
</table>

Half-adjust factors of less than .5 should not be used, since this will bring up the problem of inexact fractions.

If GROSS is negative, an 11-zone punch will be added to the code for the low-order digit. For example, if GROSS is -275869.0, the result will be 000002758P, where the character P is equivalent to a 7,11 punch.

Integer to A1

This requires two steps, since PUT operates on real numbers, not integers. If you have an integer I, which you want converted to A1 format, you must first convert it to real format:

```
X = I
```

or

```
X = FLOAT(I)
```

then use the PUT subroutine. Or, in one step:

```
CALL PUT (KGROS, 1, 10, FLOAT(I), 5.0, 1)
```

will perform this conversion.
A1 to Decimal -- A1DEC

This conversion will be needed if you have chosen to use the decimal arithmetic system of CSP. The A1 field being converted was read by FORTRAN with an A1 format, or by the overlapped I/O routines.

Suppose a card contained 1968 in columns 1 through 4, and you read it with the overlapped I/O CALL READ. It would be in A1 format, in an array KOL, one character per word:

KOL (1) contains the alphabetic 1b
KOL (2) contains the alphabetic 9b
KOL (3) contains the alphabetic 6b
KOL (4) contains the alphabetic 8b

If you want to use this value in decimal arithmetic computations, it must be converted to decimal format, one digit per word. To do this, you simply code

CALL A1DEC (KOL, 1, 4, NER)

and it will be converted, in place. Note that the A1 coding is replaced by the decimal coding. The NER is an error parameter, and will be set to the position at which it last encountered an invalid character (not 0 through 9 or a blank).

The exception to this is the last (rightmost) character, which may contain an 11 or 12 punch, indicating a sign. See the table below for allowable punches:

<table>
<thead>
<tr>
<th>Digit or character without a zone punch</th>
<th>with an 11 punch</th>
<th>with a 12 punch</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank</td>
<td>- (dash)</td>
<td>&amp;</td>
</tr>
<tr>
<td>0</td>
<td>(- zero)</td>
<td>(+ zero)</td>
</tr>
<tr>
<td>1</td>
<td>J</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>K</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>L</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>N</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>O</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>P</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>Q</td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td>R</td>
<td>I</td>
</tr>
</tbody>
</table>

Decimal to A1 -- DECA1

If you are using decimal arithmetic, you must print the answers either with a series of I1 formats, or in A1 format. The latter will be the case if you desire any editing or are using the overlapped I/O routines.

The DECA1 subroutines will perform this conversion, thus operating in reverse fashion from A1DEC.

A typical use would be

CALL DECA1 (IWORK, 6, 10, NER)

which will convert the 6th through the 10th items in the IWORK array from decimal to A1 format. The NER error parameter is present but should be of limited use, since the decimal arithmetic routines should not leave any invalid digits in the field.

The rightmost digit is assumed to carry the sign and, if negative, will be converted to the proper character plus an 11 punch.
A1 to A2 -- PACK

This conversion is very desirable if you wish to store alphabetic data on the disk. For input, output, and editing, your data must be in A1 format; however, A2 format will pack twice as much data in an equivalent number of words.

The PACK subroutine gives you the ability to convert from A1 to A2 format. For example, suppose the array IUNPK contains 123bGO:

IUNPK (1) contains an alphabetic 1
IUNPK (2) contains an alphabetic 2
IUNPK (3) contains an alphabetic 3
IUNPK (4) contains an alphabetic blank
IUNPK (5) contains an alphabetic G
IUNPK (6) contains an alphabetic O

Now suppose we say

CALL PACK (IUNPK, 1, 6, IPAKD, 1)

The data is packed and moved from IUNPK to IPAKD:

IPAKD (1) contains an alphabetic 1 and 2
IPAKD (2) contains an alphabetic 3 and blank
IPAKD (3) contains an alphabetic G and O

The IUNPK array remains unchanged. An even number of characters will be packed. Therefore, the A1 field should contain an even number of characters; otherwise, the last character in the IPAKD array will be meaningless.

A2 to A1 -- UNPAC

To convert A2 data back to A1, the UNPAC subroutine may be used. A typical call to UNPAC would be

CALL UNPAC (IPAKD, 1, 3, IUNPK, 1)

which would unpack the 123bGO packed in the previous example.
Other Code Conversions

As Figure 70.23 shows, there are other code conversions that you may require. However, since they are unusual and can be performed as a combination of several other steps, they will not be discussed in detail.
Other Character Handling Techniques

Editing Output—EDIT

Most commercial applications are strongly oriented toward the format and appearance of the output results, as opposed to the technical job, where all you want is the answer. For example:

- Dollar amounts should have commas, dollar signs, and so on.
- Invoices should show a CR symbol after negative values.
- Balance sheets should have a minus sign following a negative item.
- Punched card output should have leading zeros, so that the cards may be handled properly with a mechanical sorter.

The EDIT subroutine enables you to do all these formatting tasks. Its use requires two fields, stored in A1 format in integer arrays:

1. The source field or the field which will be edited
2. The edit mask, a field which you have coded to indicate how you want the edited output to appear.

A typical call to the EDIT subroutine is

```
CALL EDIT (KSOUR, 1, 10, MASK, 1, 13)
```

where the source field consists of items 1-10 in the KSOUR array, and the mask consists of items 1-13 in the MASK array. After editing, the MASK field is replaced by the edited source field; if you wish to use it again, therefore, you must save it somewhere else. Usually, the mask will be moved into the output area, and the source field will be edited into the output array. Thus the original mask is not destroyed. For example:

```
CALL MOVE (MASK, 1, 13, KOUT, 36)
CALL EDIT (KSOUR, 1, 10, KOUT, 36, 48)
```

Figure 70.24 is a worksheet that you may use for setting up an edit mask. The principles involved are shown best by examples (see Figures 70.25-70.30).

Moving Data Fields—MOVE

Often it becomes necessary to move the data in one array into another array—especially if you are using CSP. The MOVE subroutine has been included in CSP to facilitate such operations. Its use is quite simple, since it does no more than move data from one place to another. For example:

```
CALL MOVE (IFROM, 6, 8, ITO, 14)
```

will move

- IFROM (6) to ITO (14)
- IFROM (7) to ITO (15)
- IFROM (8) to ITO (16)

leaving the IFROM array undisturbed.

Note that the ending position in the ITO array is not supplied as one of the parameters.

The format of the data items is not affected. They may be A1, A2, decimal, or integer (but not real).
## EDIT WORKSHEET

<table>
<thead>
<tr>
<th>PROGRAM</th>
<th>PROGRAMMER</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENTS:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### STEP 1.
FILL IN LINE a, SHOWING THE LARGEST POSSIBLE SOURCE FIELD, AND WHAT YOU WANT IT TO LOOK LIKE AFTER EDITING.

**HINT:** PUT POSITION 1 OF THE SOURCE FIELD IN POSITION 2 OF THE MASK, AND SO ON, LEFT TO RIGHT.

### STEP 2.
IF YOU HAVE INSERTED ANY SPECIAL CHARACTERS INTO THE EDITED OUTPUT, PUT THEM IN THE EDIT MASK IN THE SAME POSITION IN WHICH THEY APPEAR.

**NOTE:** THIS DOES NOT APPLY TO *'s (ASTERISKS), b's (BLANKS), OR $'s (DOLLAR SIGNS). DO NOT PLACE THEM IN THE EDIT MASK YET.

**NOTE:** ALLOWABLE SPECIAL CHARACTERS ARE A THRU Z, 1 THRU 9, AND /, +, = etc.

### STEP 3.
FILL IN LINE b, SHOWING HOW YOU WANT ZERO TO APPEAR IN YOUR EDITED OUTPUT.

### STEP 4.
WHAT DID YOU DO WITH LEADING ZEROS? (YOU MAY ONLY CHOOSE ONE OPTION)

- a) LEFT THEM AS ZEROS? THEN DO NOTHING TO THE MASK.
- b) REPLACED THEM WITH ASTERISKS? IF SO, NOTE THE RIGHTMOST ASTERISK AND PUT AN ASTERISK IN THE MASK IN THE SAME POSITION.
- c) REPLACED THEM WITH BLANKS? IF SO NOTE THE RIGHTMOST BLANK AND PUT A ZERO IN THE MASK IN THE SAME POSITION.
- d) REPLACED THEM WITH A STRING OF BLANKS AND A DOLLAR SIGN? (FOR EXAMPLE bbb$b$. IF SO, NOTE THE POSITION OF THE DOLLAR SIGN AND PUT A DOLLAR SIGN IN THAT POSITION IN THE MASK.

### STEP 5.
FILL IN LINE c, SHOWING A TYPICAL NEGATIVE FIELD, AND HOW YOU WANT IT TO APPEAR.

### STEP 6.
WHAT DO YOU WANT DONE WITH A NEGATIVE FIELD INDICATOR? CHOOSE ONE.

- a) NOTHING, FIELD WILL NEVER BE NEGATIVE
- b) LETTERS 'CR' AFTER THE FIELD
- c) MINUS SIGN IN ITS OWN COLUMN, AFTER THE FIELD
- d) 11-PUNCH OVER ONE OF THE CHARACTERS

**CAUTION:** "CERTAIN ZONE PUNCHES (11, 0 AND 12, 0) CANNOT BE HANDLED BY FORTRAN I/O. IF THESE PUNCHES WILL OCCUR, YOU MUST USE CSP I/O."

### STEP 7.
HOW MANY CHARACTERS WERE IN THE FIRST SOURCE FIELD?

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
</table>

**CAUTION:** a CAN BE EQUAL TO OR LESS THAN b, BUT CANNOT BE LARGER!

### STEP 8.
DON'T FORGET; THE SOURCE FIELD MUST BE IN A1 FORMAT, WITH THE SIGN OVER THE RIGHTMOST CHARACTER.

### SOURCE FIELD

| a | b | c |

### DESIRED EDITED OUTPUT

| a | b | c |

**IMPLIED SIGN**

| REQUIRED EDIT MASK |

---

Figure 70, 24,
EDIT WORKSHEET

PROGRAM	 PROGRAMMER	 DATE

COMMENTS: MONETARY FIELD, TO BE PUNCHED INTO CARD WITH LEADING ZEROS DESIRED, BUT NO COMMAS OR DEC PT. IF PUNCHED OVER RIGHTMOST (UNITS) POSITION IF NEGATIVE.

STEP 1. FILL IN LINE a, SHOWING THE LARGEST POSSIBLE SOURCE FIELD, AND WHAT YOU WANT IT TO LOOK LIKE AFTER EDITING. HINT: PUT POSITION 1 OF THE SOURCE FIELD IN POSITION 2 OF THE MASK, AND SO ON, LEFT TO RIGHT.

STEP 2. IF YOU HAVE INSERTED ANY SPECIAL CHARACTERS INTO THE EDITED OUTPUT, PUT THEM IN THE EDIT MASK IN THE SAME POSITION IN WHICH THEY APPEAR.

NOTE: THIS DOES NOT APPLY TO "*" (ASTERISKS), "i" (BLANKS), OR "$" (DOLLAR SIGNS). DO NOT PLACE THEM IN THE EDIT MASK YET.

NOTE: ALLOWABLE SPECIAL CHARACTERS ARE A THRU Z, 1 THRU 9, AND /, -+= etc.

STEP 3. FILL IN LINE b, SHOWING HOW YOU WANT ZERO TO APPEAR IN YOUR EDITED OUTPUT.

STEP 4. WHAT DID YOU DO WITH LEADING ZEROS? (YOU MAY ONLY CHOOSE ONE OPTION)
   a) LEFT THEM AS ZEROS? THEN DO NOTHING TO THE MASK.
   b) REPLACED THEM WITH ASTERISKS? IF SO, NOTE THE RIGHTMOST ASTERISK AND PUT AN ASTERISK IN THE MASK IN THE SAME POSITION.
   c) REPLACED THEM WITH BLANKS? IF SO NOTE THE RIGHTMOST BLANK AND PUT A ZER0 IN THE MASK IN THE SAME POSITION.
   d) REPLACED THEM WITH A STRING OF BLANKS AND A DOLLAR SIGN? (FOR EXAMPLE bbbbb$). IF SO, NOTE THE POSITION OF THE DOLLAR SIGN AND PUT A DOLLAR SIGN IN THAT POSITION IN THE MASK.

STEP 5. FILL IN LINE c, SHOWING A TYPICAL NEGATIVE FIELD, AND HOW YOU WANT IT TO APPEAR.

STEP 6. WHAT DO YOU WANT DONE WITH A NEGATIVE FIELD INDICATOR? CHOOSE ONE.
   a) NOTHING, FIELD WILL NEVER BE NEGATIVE. DO NOTHING.
   b) LETTERS 'CR' AFTER THE FIELD PUT A 'CR' IN THE MASK TO THE RIGHT OF THE FIELD.
   c) MINUS SIGN IN ITS OWN COLUMN, AFTER THE FIELD PUT A MINUS SIGN IN THE POSITION RIGHT AFTER THE FIELD.
   d) 11-PUNCH OVER ONE OF THE CHARACTERS SAME AS OPTION C, THEN USE NZONE SUBROUTINE TO MOVE ZONE PUNCH TO THE DESIRED POSITION.

CAUTION: "CERTAIN ZONE PUNCHES (11, 0 AND 12, 0) CANNOT BE HANDLED BY FORTRAN I/O. IF THESE PUNCHES OCCUR, YOU MUST USE C8P I/O."

STEP 7. HOW MANY CHARACTERS WERE IN THE FIRST SOURCE FIELD? 9
HOWN MANY BLANKS REMAIN IN THE MASK? 10
CAUTION: a CAN BE EQUAL TO OR LESS THAN b, BUT CANNOT BE LARGER!

STEP 8. DON'T FORGET; THE SOURCE FIELD MUST BE IN A1 FORMAT, WITH THE SIGN OVER THE RIGHTMOST CHARACTER.

<table>
<thead>
<tr>
<th>SOURCE FIELD</th>
<th>DESIRED EDITED OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>LINE a – LARGEST</td>
</tr>
<tr>
<td>b</td>
<td>LINE b – ZERO</td>
</tr>
<tr>
<td>c</td>
<td>LINE c – TYPICAL NEGATIVE</td>
</tr>
<tr>
<td>IMPLIED SIGN</td>
<td>REQUIRED EDIT MASK</td>
</tr>
</tbody>
</table>

Figure 70, 25.
## EDIT WORKSHEET

**PROGRAM**

**PROGRAMMER**

**DATE**

**COMMENTS:** MONETARY FIELD, WITH FLOATING $, AND NEGATIVE INDICATOR (A - IN COLUMN FOLLOWING).

### STEP 1.
Fill in line a, showing the largest possible source field, and what you want it to look like after editing.

**HINT:** Put position 1 of the source field in position 2 of the mask, and so on, left to right.

### STEP 2.
If you have inserted any special characters into the edited output, put them in the edit mask in the same position in which they appear.

**NOTE:** This does not apply to "1" (asterisks), b's (blanks), or $'s (dollar signs). Do not place them in the edit mask yet.

**NOTE:** Allowable special characters are a thru z, 1 thru 9, and /, - + .. etc.

### STEP 3.
Fill in line b, showing how you want zero to appear in your edited output.

### STEP 4.
What did you do with leading zeros? (You may only choose one option)

- **a)** Left them as zeros? Then do nothing to the mask.
- **b)** Replaced them with asterisks? If so, note the rightmost asterisk and put an asterisk in the mask in the same position.
- **c)** Replaced them with blanks? If so, note the rightmost blank and put a zero in the mask in the same position.
- **d)** Replaced them with a string of blanks and a dollar sign? (for example b b b b $1). If so, note the position of the dollar sign and put a dollar sign in that position in the mask.

### STEP 5.
Fill in line c, showing a typical negative field, and how you want it to appear.

### STEP 6.
What do you want done with a negative field indicator? Choose one.

- **a)** Nothing, field will never be negative. Do nothing.
- **b)** Letters 'CR' after the field. Put a 'CR' in the mask to the right of the field.
- **c)** Minus sign in its own column, after the field. Put a minus sign in the position right after the field.
- **d)** 11-punch over one of the characters. Same as option c, then use nzone subroutine to move zone punch to the desired position.

**CAUTION:** "Certain zone punches (11, 0 and 12,0) cannot be handled by FORTRAN I/O. If these punches will occur, you must use CSP I/O."

### CALL NZONE (MASK, 11, NOLDZ)

### MOVE ZONE FROM HERE TO HERE

### CALL NZONE (MASK, 9, NOLDZ, JUNK)

### STEP 7.
How many characters were in the first source field?...

**a)** How many blanks remain in the mask?...

**CAUTION:** a can be equal to or less than b, but cannot be larger.

### STEP 8.
Don't forget; the source field must be in A1 format, with the sign over the rightmost character.

<table>
<thead>
<tr>
<th>SOURCE FIELD</th>
<th>DESIRED EDITED OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>g g g g g g g g g g g</td>
</tr>
<tr>
<td>b</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>c</td>
<td>1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

**IMPLIED SIGN**

**REQUIRED EDIT MASK**

---

Figure 70, 26,
EDIT WORKSHEET

**STEP 1.** Fill in Line a, showing the largest possible source field, and what you want it to look like after editing. Hint: Put position 1 of the source field in position 2 of the mask, and so on, left to right.

**STEP 2.** If you have inserted any special characters into the edited output, put them in the edit mask in the same position in which they appear.

Note: This does not apply to "*" (asterisks), "'" (blanks), or "$" (dollar signs). Do not place them in the edit mask yet.

Note: Allowable special characters are A thru Z, 1 thru 9, and /, - + = etc.

**STEP 3.** Fill in Line b, showing how you want zero to appear in your edited output.

**STEP 4.** What did you do with leading zeros? (You may only choose one option)
   a) Left them as zeros? Then do nothing to the mask.
   b) Replaced them with asterisks? If so, note the rightmost asterisk and put an asterisk in the mask in the same position.
   c) Replaced them with blanks? If so, note the rightmost blank and put a zero in the mask in the same position.
   d) Replaced them with a string of blanks and a dollar sign? (For example bbbb$). If so, note the position of the dollar sign and put a dollar sign in that position in the mask.

**STEP 5.** Fill in Line c, showing a typical negative field, and how you want it to appear.

**STEP 6.** What do you want done with a negative field indicator? Choose one.
   a) Nothing, field will never be negative. Do nothing.
   b) Letters 'CR' after the field. Put a 'CR' in the mask to the right of the field.
   c) Minus sign in its own column, after the field. Put a minus sign in the position right after the field.
   d) 11-punch over one of the characters. Same as option c, then use NZONE subroutine to move zone punch to the desired position.

Caution: "Certain zone punches (11, 0 and 12, 0) cannot be handled by Fortran I/O. If these punches will occur, you must use CSP I/O."

**STEP 7.** How many characters were in the first source field? 9

How many blanks remain in the mask? 9

Caution: a can be equal to or less than b, but **cannot** be larger!

**STEP 8.** Don't forget; the source field must be in A1 format, with the sign over the rightmost character.

Figure 70.27.
**EDIT WORKSHEET**

### Comments

**PROGRAM**

**PROGRAMMER**

**DATE**

---

**STEP 1.** FILL IN LINE a, SHOWING THE LARGEST POSSIBLE SOURCE FIELD, AND WHAT YOU WANT IT TO LOOK LIKE AFTER EDITING.

**HINT:** PUT POSITION 1 OF THE SOURCE FIELD IN POSITION 2 OF THE MASK, AND SO ON, LEFT TO RIGHT.

**STEP 2.** IF YOU HAVE INSERTED ANY SPECIAL CHARACTERS INTO THE EDITED OUTPUT, PUT THEM IN THE Edit MASK IN THE SAME POSITION IN WHICH THEY APPEAR.

**NOTE:** THIS DOES NOT APPLY TO "s (ASTERISKS), b's (BLANKS), OR $'s (DOLLAR SIGNS). DO NOT PLACE THEM IN THE EDIT MASK YET.

**STEP 3.** FILL IN LINE b, SHOWING HOW YOU WANT ZERO TO APPEAR IN YOUR EDITED OUTPUT.

**STEP 4.** WHAT DID YOU DO WITH LEADING ZEROS? (YOU MAY ONLY CHOOSE ONE OPTION)

a) LEFT THEM AS ZEROS? THEN DO NOTHING TO THE MASK.

b) REPLACED THEM WITH ASTERISKS? IF SO, NOTE THE RIGHTMOST ASTERISK AND PUT AN ASTERISK IN THE MASK IN THE SAME POSITION.

c) REPLACED THEM WITH BLANKS? IF SO, NOTE THE RIGHTMOST BLANK AND PUT A ZERO IN THE MASK IN THE SAME POSITION.

d) REPLACED THEM WITH A STRING OF BLANKS AND A DOLLAR SIGN? (FOR EXAMPLE bbbb), IF SO, NOTE THE POSITION OF THE DOLLAR SIGN AND PUT A DOLLAR SIGN IN THAT POSITION IN THE MASK.

**STEP 5.** FILL IN LINE c, SHOWING A TYPICAL NEGATIVE FIELD, AND HOW YOU WANT IT TO APPEAR.

**STEP 6.** WHAT DO YOU WANT DONE WITH A NEGATIVE FIELD INDICATOR? CHOOSE ONE.

a) NOTHING, FIELD WILL NEVER BE NEGATIVE

b) LETTERS 'CR' AFTER THE FIELD

c) MINUS SIGN IN ITS OWN COLUMN AFTER THE FIELD

d) 11-PUNCH OVER ONE OF THE CHARACTERS

**CAUTION:** "CERTAIN ZONE PUNCHES 111, 0 AND CALL NZONE (MASK, E, NOLDZ, JUNK)

**STEP 7.** HOW MANY CHARACTERS WERE IN THE FIRST SOURCE FIELD?...

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th></th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

**CAUTION:** a CAN BE EQUAL TO OR LESS THAN b, BUT CANNOT BE LARGER!

**STEP 8.** DON'T FORGET: THE SOURCE FIELD MUST BE IN A1 FORMAT, WITH THE SIGN OVER THE RIGHTMOST CHARACTER.

---

**SOURCE FIELD**

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

**DESIRED EDITED OUTPUT**

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

**IMPLIED SIGN**

**REQUIRED EDIT MASK**

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
<td>b</td>
<td>6</td>
<td>b</td>
<td>6</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Figure 70, 28.
## EDIT WORKSHEET

**COMMENTS:** DATE

### STEP 1.
**FILL IN LINE a, SHOWING THE LARGEST POSSIBLE SOURCE FIELD, AND WHAT YOU WANT IT TO LOOK LIKE AFTER EDITING.**
**HINT:** PUT POSITION 1 OF THE SOURCE FIELD IN POSITION 2 OF THE MASK, AND SO ON, LEFT TO RIGHT.

### STEP 2.
**IF YOU HAVE INSERTED ANY SPECIAL CHARACTERS INTO THE EDITED OUTPUT, PUT THEM IN THE EDIT MASK IN THE SAME POSITION IN WHICH THEY APPEAR.**
- **NOTE:** THIS DOES NOT APPLY TO "s (ASTERISKS), b's (BLANKS), OR $'s (DOLLAR SIGNS). DO NOT PLACE THEM IN THE EDIT MASK YET.
- **NOTE:** ALLOWABLE SPECIAL CHARACTERS ARE A THRU Z, 1 THRU 9, AND /, -, + = etc.

### STEP 3.
**FILL IN LINE b, SHOWING HOW YOU WANT ZERO TO APPEAR IN YOUR EDITED OUTPUT.**

### STEP 4.
**WHAT DID YOU DO WITH LEADING ZEROS? (YOU MAY ONLY CHOOSE ONE OPTION)**
- a) LEFT THEM AS ZEROS? THEN DO NOTHING TO THE MASK.
- b) REPLACED THEM WITHASTERISKS? IF SO, NOTE THE RIGHTMOST ASTERISK AND PUT AN ASTERISK IN THE MASK IN THE SAME POSITION.
- c) REPLACED THEM WITH BLANKS? IF SO NOTE THE RIGHTMOST BLANK AND PUT A ZERO IN THE MASK IN THE SAME POSITION.
- d) REPLACED THEM WITH A STRING OF BLANKS AND A DOLLAR SIGN? (FOR EXAMPLE bbbb$). IF SO, NOTE THE POSITION OF THE DOLLAR SIGN AND PUT A DOLLAR SIGN IN THAT POSITION IN THE MASK.

### STEP 5.
**FILL IN LINE c, SHOWING A TYPICAL NEGATIVE FIELD, AND HOW YOU WANT IT TO APPEAR.**

### STEP 6.
**WHAT DO YOU WANT DONE WITH A NEGATIVE FIELD INDICATOR?** CHOOSE ONE.
- a) NOTHING, FIELD WILL NEVER BE NEGATIVE, DO NOTHING.
- b) LETTERS 'CR' AFTER THE FIELD, PUT A 'CR' IN THE MASK TO THE RIGHT OF THE FIELD.
- c) MINUS SIGN IN ITS OWN COLUMN, AFTER THE FIELD, PUT A MINUS SIGN IN THE POSITION RIGHT AFTER THE FIELD.
- d) 11-PUNCH OVER ONE OF THE CHARACTERS, SAME AS OPTION C, THEN USE NZONE SUBROUTINE TO MOVE ZONE PUNCH TO THE DESIRED POSITION.' CAUTION: "CERTAIN ZONE PUNCHES (11, 0 AND 12, 0) CANNOT BE HANDLED BY FORTRAN I/O. IF THESE PUNCHES WILL OCCUR, YOU MUST USE CSP I/O."

### STEP 7.
**HOW MANY CHARACTERS WERE IN THE FIRST SOURCE FIELD?...**
- a) 5
- b) 5
**CAUTION:** a CAN BE EQUAL TO OR LESS THAN b, BUT CANNOT BE LARGER!

### STEP 8.
**DON'T FORGET; THE SOURCE FIELD MUST BE IN A1 FORMAT, WITH THE SIGN OVER THE RIGHTMOST CHARACTER.**

<table>
<thead>
<tr>
<th>SOURCE FIELD</th>
<th>DESIRED EDITED OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 0 6 7</td>
<td>a 1 2 0 6 7</td>
</tr>
<tr>
<td>b</td>
<td>a 1 2 0 6 7</td>
</tr>
<tr>
<td>c</td>
<td>a 1 2 0 6 7</td>
</tr>
<tr>
<td>IMPLIED SIGN</td>
<td></td>
</tr>
<tr>
<td>REQUIRED EDIT MASK</td>
<td>1 2 0 6 7</td>
</tr>
</tbody>
</table>

Figure 70, 29.
EDIT WORKSHEET

PROGRAM	 PROGRAMMER	 DATE

COMMENTS: MONETARY FIELD, WITH CR SYMBOL, LEADING *

STEP 1. FILL IN LINE a, SHOWING THE LARGEST POSSIBLE SOURCE FIELD, AND WHAT YOU WANT IT TO LOOK LIKE AFTER EDITING.
HINT: PUT POSITION 1 OF THE SOURCE FIELD IN POSITION 2 OF THE MASK, AND SO ON, LEFT TO RIGHT.

STEP 2. IF YOU HAVE INSERTED ANY SPECIAL CHARACTERS INTO THE EDITED OUTPUT, PUT THEM IN THE EDIT MASK IN THE SAME POSITION IN WHICH THEY APPEAR.
NOTE: THIS DOES NOT APPLY TO **s (ASTERISKS), b's (BLANKS), OR $s (DOLLAR SIGNS). DO NOT PLACE THEM IN THE EDIT MASK YET.

STEP 3. FILL IN LINE b, SHOWING HOW YOU WANT ZERO TO APPEAR IN YOUR EDITED OUTPUT.

STEP 4. WHAT DID YOU DO WITH LEADING ZEROS? (YOU MAY ONLY CHOOSE ONE OPTION)
a) LEFT THEM AS ZEROS THEN DO NOTHING TO THE MASK.
b) REPLACED THEM WITH ASTERISKS? IF SO, NOTE THE RIGHTMOST ASTERISK AND PUT AN ASTERISK IN THE MASK IN THE SAME POSITION.
c) REPLACED THEM WITH BLANKS? IF SO NOTE THE RIGHTMOST BLANK AND PUT A ZERO IN THE MASK IN THE SAME POSITION.
d) REPLACED THEM WITH A STRING OF BLANKS AND A DOLLAR SIGN? (FOR EXAMPLE bbb$b). IF SO, NOTE THE POSITION OF THE DOLLAR SIGN AND PUT A DOLLAR SIGN IN THAT POSITION IN THE MASK.

STEP 5. FILL IN LINE c, SHOWING A TYPICAL NEGATIVE FIELD, AND HOW YOU WANT IT TO APPEAR.

STEP 6. WHAT DO YOU WANT DONE WITH A NEGATIVE FIELD INDICATOR? CHOOSE ONE
a) NOTHING, FIELD WILL NEVER BE NEGATIVE. DO NOTHING.
b) LETTERS 'CR' AFTER THE FIELD. PUT A 'CR' IN THE MASK TO THE RIGHT OF THE FIELD.
c) MINUS SIGN IN ITS OWN COLUMN AFTER THE FIELD. PUT A MINUS SIGN IN THE POSITION RIGHT AFTER THE FIELD.
d) 11-PUNCH OVER ONE OF THE CHARACTERS. SAME AS OPTION c, THEN USE NZONE SUBROUTINE TO MOVE ZONE PUNCH TO THE DESIRED POSITION.

CAUTION: "CERTAIN ZONE PUNCHES (11, 0 AND 12, 0) CANNOT BE HANDLED BY FORTRAN I/O. IF THESE PUNCHES WILL OCCUR, YOU MUST USE CSP I/O."

CALL NZONE (MASK, 5, NOLDZ)
MOVE ZONE FROM HERE TO HERE
CALL NZONE (MASK, NOLDZ, JUNK)

STEP 7. HOW MANY CHARACTERS WERE IN THE FIRST SOURCE FIELD?...

STEP 8. DON'T FORGET; THE SOURCE FIELD MUST BE IN A1 FORMAT, WITH THE SIGN OVER THE RIGHTMOST CHARACTER.

SOURCE FIELD

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 9 9 9 9 9 9 9 9 9 9 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2 3 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IMPLIES SIGN

REQUIRED EDIT MASK

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 6 6 6 6 6 6 6 6 6 6 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 3 4 CR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 70, 30.
Filling a Field with a Specific Character--FILL

If your program requires that you create long strings of the same digit or character, the FILL subroutine may be used. The statement

```
CALL FILL (KARRY, 10, 36, IT)
```

will place the coding of IT in positions 10-36 of the array KARRY. IT may be any integer between +32767 and -32768.

In a standard FORTRAN program, this is a useful way to clear a set of totals to zero.

If you are using the decimal arithmetic routines, this can also be used to clear a total field to zero.

When using the overlapped I/O routines, it is often necessary to fill an area with blanks, dashes, or some other character.

A table of the decimal equivalent of various EBCDIC (A1) characters may be found in the CSP manual. However, it is usually easier to obtain their value indirectly with a DATA statement. For example, to fill a printer output line with dashes, you would place a DATA statement in the beginning of your program:

```
DATA IDASH/ ' - '/
```

placing the dash character between the quotes or apostrophes. Then the FILL statement

```
CALL FILL (IOUT, 1, 120, IDASH)
```

would fill the IOUT array with the A1 code for a dash.

Comparing Alphabetic Fields--NCOMP

The requirements for alphabetic comparisons can usually be broken into two main classifications:

1. Comparing to determine whether there is a match/no match condition.
2. Comparing to determine whether one field is higher than, lower than or equal to another field.

Because the first is quite a bit simpler than the second, these two types of alphabetic compares will be discussed separately.
Match/No Match Alpha Compare. This operation is common to many commercial applications:

- An employee time card may contain a four-letter code describing what job he worked on, and the program must look up a corresponding rate.
- An inventory card may contain a two-letter code indicating unit of measure—LB, GR, EA, etc.
- The name field on each input card is compared with the name field on the preceding card; if they are not the same, branch to the "control break" section of the program.

If the fields to be compared are one or two characters long, they may be read into a single integer variable and compared like any other integers. For example, if their names are ITHIS and ITHAT, the statement:

\[
\text{IF}(\text{ITHIS}-\text{ITHAT}) \ 1, \ 2, \ 1
\]

will branch to statement number 2 if they are identical, and statement number 1 if they are different. The format (A1 or A2) does not matter, except, of course, that it must be the same for both.

If the fields are longer than two characters, they should be read into integer arrays, in A1 format, and compared with the NCOMP function. Using the previous example, suppose ITHIS and ITHAT are arrays, each containing ten alphabetic characters.

\[
\text{IF}(\text{NCOMP}(\text{ITHIS}, 1, 10, \text{ITHAT}, 1)) \ 1, \ 2, \ 1
\]

will work the same as the simple IF statement shown earlier.

Don't try to compare alphabetic fields that have been stored as real variables. Two six-character fields, called THIS and THAT, may be read from a card and moved about in core just like any other real variables; however, they cannot be compared validly. The statement

\[
\text{IF}(\text{THIS}-\text{THAT}) \ 1, \ 2, \ 1
\]

will not always branch to statement number 1 if the two fields are different.
High/Low/Equal Alpha Compare. Everything said about the Match/No Match compare also applies here, with two exceptions:

1. The fields to be compared should always be in A1 format.
2. The A1 representation for a blank must be changed if you want it to fall in the proper collating sequence.

Figure 70.31 shows the decimal representation of various characters in A1 format. Note that the blank falls after the letters and numbers. If it is left there, alphabetic compares will yield an ascending sequence—for example:

WILLIAMSON
WILLIAMSbb
WILLIAMbbb

rather than the correct

WILLIAMbbb
WILLIAMSbb
WILLIAMSON

This can easily be corrected if blanks are converted from 16448 to something less than -16064, the letter A. In fact, you might as well change it to -16448. With a DO loop, the input record can be scanned for +16448s, and each one found can be changed to -16448.

They need not be converted back to +16448 for printed output, since any invalid character (such as -16448) will be printed as a blank anyway. For punched output, however, this will not be so, and the -16448s should be changed back to +16448s.

<table>
<thead>
<tr>
<th>Character</th>
<th>A1 Equivalent</th>
<th>Character</th>
<th>A1 Equivalent</th>
<th>Character</th>
<th>A1 Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-16064</td>
<td>S</td>
<td>-7616</td>
<td>blank</td>
<td>16448</td>
</tr>
<tr>
<td>B</td>
<td>-15808</td>
<td>T</td>
<td>-7360</td>
<td>. (period)</td>
<td>19264</td>
</tr>
<tr>
<td>C</td>
<td>-15552</td>
<td>U</td>
<td>-7104</td>
<td>&lt;= (less than)</td>
<td>19520</td>
</tr>
<tr>
<td>D</td>
<td>-15296</td>
<td>V</td>
<td>-6848</td>
<td>(</td>
<td>19776</td>
</tr>
<tr>
<td>E</td>
<td>-15040</td>
<td>W</td>
<td>-6592</td>
<td>+</td>
<td>20032</td>
</tr>
<tr>
<td>F</td>
<td>-14784</td>
<td>X</td>
<td>-6336</td>
<td>&amp;</td>
<td>20544</td>
</tr>
<tr>
<td>G</td>
<td>-14528</td>
<td>Y</td>
<td>-6080</td>
<td>$</td>
<td>23360</td>
</tr>
<tr>
<td>H</td>
<td>-14272</td>
<td>Z</td>
<td>-5824</td>
<td>*</td>
<td>23616</td>
</tr>
<tr>
<td>I</td>
<td>-14016</td>
<td>0</td>
<td>-4032</td>
<td>)</td>
<td>23872</td>
</tr>
<tr>
<td>J</td>
<td>-11968</td>
<td>1</td>
<td>-3776</td>
<td>-(minus)</td>
<td>24640</td>
</tr>
<tr>
<td>K</td>
<td>-11712</td>
<td>2</td>
<td>-3520</td>
<td>/</td>
<td>24896</td>
</tr>
<tr>
<td>L</td>
<td>-11466</td>
<td>3</td>
<td>-3264</td>
<td>,</td>
<td>27456</td>
</tr>
<tr>
<td>M</td>
<td>-11200</td>
<td>4</td>
<td>-3008</td>
<td>%</td>
<td>27712</td>
</tr>
<tr>
<td>N</td>
<td>-10944</td>
<td>5</td>
<td>-2752</td>
<td>#</td>
<td>31552</td>
</tr>
<tr>
<td>O</td>
<td>-10688</td>
<td>6</td>
<td>-2496</td>
<td>@</td>
<td>31908</td>
</tr>
<tr>
<td>P</td>
<td>-10432</td>
<td>7</td>
<td>-2240</td>
<td>' (apostrophe)</td>
<td>32064</td>
</tr>
<tr>
<td>Q</td>
<td>-10176</td>
<td>8</td>
<td>-1984</td>
<td>=</td>
<td>32320</td>
</tr>
<tr>
<td>R</td>
<td>-9920</td>
<td>9</td>
<td>-1728</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note position of blank.
Working with Zone Punches -- NZONE

The top three rows of the data processing card are commonly called the "zone" rows, and a punch in one of them is called a zone punch. The top row is called the 12 zone; the next, the 11 zone; and the next (the 0 row), the 0 zone. (See Figure 70.32.) A digit overpunched with a 12 zone punch is taken to be positive; an 11 punch indicates negative. This is quite reasonable, since an 11 punch alone is a minus sign, and a 12 punch is an ampersand (&) or plus sign (+), depending on the coding scheme and cardpunch used. (While many people use the term "X punch" instead of 11 punch, both mean the same.) The 12 punch is rarely used, since it is easier to have no zone punch for positive numbers.

The zone punch, when used to indicate the sign, should be placed over the units (rightmost) position of the field. For example, -1675 would be punched with an 11 punch over the 5.

This practice will result in a card code equivalent to one of the letters J through R, or a negative zero. The table below shows the card code equivalents:

<table>
<thead>
<tr>
<th>These punches</th>
<th>Mean either this</th>
<th>Or this</th>
</tr>
</thead>
<tbody>
<tr>
<td>11, 0</td>
<td>-0</td>
<td>Æ</td>
</tr>
<tr>
<td>11, 1</td>
<td>-1</td>
<td>J</td>
</tr>
<tr>
<td>11, 2</td>
<td>-2</td>
<td>K</td>
</tr>
<tr>
<td>11, 3</td>
<td>-3</td>
<td>L</td>
</tr>
<tr>
<td>11, 4</td>
<td>-4</td>
<td>M</td>
</tr>
<tr>
<td>11, 5</td>
<td>-5</td>
<td>N</td>
</tr>
<tr>
<td>11, 6</td>
<td>-6</td>
<td>O</td>
</tr>
<tr>
<td>11, 7</td>
<td>-7</td>
<td>P</td>
</tr>
<tr>
<td>11, 8</td>
<td>-8</td>
<td>Q</td>
</tr>
<tr>
<td>11, 9</td>
<td>-9</td>
<td>R</td>
</tr>
</tbody>
</table>

If the card containing the 1675 field were interpreted, or listed character for character, it would appear as 167N, where the character N is equivalent to a 5 and an 11 punch.

In a few cases, zone punches may also be found in card columns other than the low-order digits of a numeric field. This is particularly true in installations that once had a unit record, or punched card, system. In such a system, zone punches provided an easy way to pack additional data into a punched card.

One of the advantages of the CSP overlapped I/O routines is that they allow the input and output of fields with zone punches. This is normally quite difficult with standard FORTRAN READs and WRITEs, since the 11,0 (~ zero) punch is not permitted.

---

Figure 70.32.
The NZONE Subroutine. The NZONE subroutine has been included in CSP to allow you to interrogate a zone punch, obtaining a code that indicates its status, and to modify a zone punch. If you wished to operate on the 18th character in the INOUT array, the call to NZONE would be

```
CALL NZONE (INOUT, 18, NEWZ, NOLDZ)
```

NOLDZ will be returned to you, indicating what zone punch was present:

<table>
<thead>
<tr>
<th>NOLDZ</th>
<th>Zone Punch</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>A--I</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>J--R</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>S--Z</td>
</tr>
<tr>
<td>4</td>
<td>none</td>
<td>0--9</td>
</tr>
<tr>
<td>more than 4</td>
<td></td>
<td>special character</td>
</tr>
</tbody>
</table>

Note that an NOLDZ of 4 or more does not tell you what zone punch was present, but only that INOUT(18) contains a special character.

You supply the NEWZ parameter, indicating to the subroutine what you want done with the old zone punch:

<table>
<thead>
<tr>
<th>NEWZ</th>
<th>Action Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Make the zone a 12</td>
</tr>
<tr>
<td>2</td>
<td>Make the zone an 11</td>
</tr>
<tr>
<td>3</td>
<td>Make the zone a 0</td>
</tr>
<tr>
<td>4</td>
<td>Remove the zone</td>
</tr>
<tr>
<td>more than 4</td>
<td>Let the zone alone</td>
</tr>
</tbody>
</table>
FORTRAN CORE SAVING TIPS

General

The way in which you code your FORTRAN programs will have a considerable effect on their size. The difference between efficient and inefficient coding might be as much as several hundred words. This may mean the difference between a program that fits in core and one that doesn't, or the difference between one that requires many time-consuming overlays and one that requires none.

In general, the larger a program, the more slowly it will run—not because it does more, but because of the overlays (SOCALs, LOCALs and LINKs) required to fit it into core. When writing your programs, therefore, you should make every effort to keep them small. One way to do this is to know which FORTRAN techniques save core storage, and which ones consume it excessively. A better way is to design programs that do just one job, rather than many. (Subsection 25.30.20 contains a discussion of the advantages of modular programming.) Still another way is to use efficient overlays (see Section 65).

The core storage requirements for any particular program can be split into three major elements:

- The object code generated by the compiler
- The subroutines, which actually do the work
- The data area, where all variables and constants are stored

You should realize that very little actual work is done "in line" by your program; when the end-of-compilation summary says your program size is 1000 words, it means that your program has been translated into 1000 words of branches or linkages to subroutines, plus some housekeeping to prepare the linkages. The exception to this statement is integer arithmetic, which is done in line, without subroutines. However, all subscript calculation, real arithmetic, and input/output is accomplished by subroutines.

Some of the core saving tips in this section are directed toward reducing the subroutine requirements, while others will reduce the amount of in-line coding.

If you modify an existing program, incorporating some of these tips, don't expect to find all the savings reflected in the end-of-compilation summary. Check the list of required subroutines; you may have eliminated some of them.
Reduction Program Size

Use the DATA Statement

The DATA statement is a recent addition to 1130 FORTRAN, having been incorporated into Version 2 of the 1130 Monitor System. Basically, it is used to create constants at the time the program is compiled, rather than each time the program is executed. It saves some time, but this should not be enough to notice in the overall run time of most programs. Much more important, the DATA statement saves core storage.

It is a nonexecutable statement, like the TYPE, DIMENSION, EQUIVALENCE, etc., statements, and requires no core storage. It provides only a starting value for variables. For exact rules concerning the use of the DATA statement, see the 1130 FORTRAN manual; in this section you will see some examples of its use.

- Case 1: Initialize Tables at the Beginning of a Program

Almost every program begins with statements such as

\[
\begin{align*}
\text{DO 16 J = 1,50} \\
\text{TOT(J) = 0.0} \\
16 \text{ SUBT(J) = 0.0}
\end{align*}
\]

This coding, which requires about 27 words of storage, can be replaced with

\[
\text{DATA TOT/50*0.0/, SUBT/50*0.0/}
\]

which requires no storage.

Let us stress three facts at this point:

1. You still require two 50-position arrays.
2. If you say TOT (1) = 1.5 later in the program, this will be done, and TOT (1) will no longer be 0.0.
3. If you want to clear out these tables again during the execution of the program, you must use the conventional DO loop. You cannot GO TO or reexecute the DATA statement, since it is a nonexecutable statement and, in fact, no longer exists once the program is loaded.

- Case 2: Initialize Indicators, etc.

The program PAY04, listed in Subsection 70.50.30, contains the following FORTRAN statements:

\[
\begin{align*}
T & = 0. \\
\text{IERR} & = 0 \\
\text{ICOL} & = 1 \\
\text{IN} & = 1 \\
\text{XOT} & = 0. \\
\text{XBN} & = 0. \\
\text{XSP} & = 0. \\
\text{XREG} & = 0. \\
\text{IPAGE} & = 0 \\
\text{LINE} & = 50
\end{align*}
\]

which require 40 words of object coding. They may be replaced by

\[
\text{DATA T/0./, IERR/0/, ICOL/1/, etc., etc.}
\]

- Case 3: Setting a Variable to Different Values

Again inspecting the same program, PAY04, we find

\[
\begin{align*}
\text{GO TO (76,77,78,79,80,81), NOPLT} \\
76 \text{ ILST}=250 \\
77 \text{ ILST}=90 \\
78 \text{ ILST}=200 \\
79 \text{ ILST}=50 \\
80 \text{ ILST}=150 \\
81 \text{ ILST}=30 \\
83 \text{ continue}
\end{align*}
\]

which requires 44 words of object coding. It may be replaced by a combination of

\[
\begin{align*}
\text{DIMENSION IFACT (6)} \\
\text{DATA IFACT/250, 90, 200, 50, 150, 30/}
\end{align*}
\]

and the statement (using eight words)

\[
\text{ILST = IFACT(NOPLT)}
\]

Placing the six constants in the IFACT array adds no core requirements, since they were in core before, as INTEGER CONSTANTS (see listing at end of compilation).
Case 4: Creating Alphabetic Masks for the EDIT Subroutine

If you want to print or punch your FORTRAN results with commas, floating dollar signs, etc., you are probably using the EDIT subroutine found in CSP. This subroutine requires an Edit mask, which may look like

```
bb, bb$. bbCR
```

There are two ways to obtain this mask, which must be in Al format in an eleven-word integer array (call it MASK). You can read it off a card, or you can look up the decimal equivalents of the EBCDIC codes, and set each one equal to the desired character:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16448</td>
<td>blank</td>
<td>16448</td>
<td>blank</td>
<td>27456</td>
<td>comma</td>
<td>16448</td>
<td>blank</td>
<td>16448</td>
<td>blank</td>
<td>23360</td>
<td>dollar sign</td>
<td>19264</td>
<td>period</td>
<td>16448</td>
<td>blank</td>
<td>16448</td>
<td>blank</td>
<td>-15552</td>
<td>letter C</td>
</tr>
</tbody>
</table>

The DATA statement allows you to eliminate this sixty-six-word series of commands, replacing it with

```
DATA MASK/'b','b',' ','b','$',' ','b','b','C','R'/
```

where b indicates a blank.

Keep FORMAT Statements Compact

1130 FORTRAN includes a very flexible repertoire of FORMAT codes, and often gives you several different ways to achieve the same results. For example, you can specify either (F6.2, F6.2, F6.2) or (3F6.2). With alphabetic heading data, there are more options. To type a line which reads

```
bbbbbbbbTOTAL
```

you can use as FORMAT statements the following:

a. FORMAT(14HbbbbbbbbTOTAL)
b. FORMAT('bbbbbbbbTOTAL')
c. FORMAT(9X,'TOTAL')
d. FORMAT(9X,5HTOTAL)
e etc.

If you suspected that some options used more core storage than others, you would be correct. Options a and b force the compiler to allocate nine words for this FORMAT STATEMENT; options c and d only require six words.

The main difference between the two styles is the manner in which you have generated nine blank columns -- 9X or 'bbbbbbbb'. The 9X is coded and compressed into one word; the 'bbbbbbbb' requires one word, plus a string of five words, each containing two alphabetic blanks.

The difference does not appear to be great, but consider your typical commercial report writing program with its many long FORMAT statements. The difference between the best (smallest core requirement) and what the programmer has actually used may be substantial.

This topic is further complicated by the fact that the X specification is best for large numbers of spaces, while the literal or ' specification is best for small numbers. In summary, to get one or two spaces, it is best to enclose blanks within quotes (or use the H specification). To get three or more spaces, use the X specification.
Code Efficient I/O Statements

The manner in which you code your I/O statements can have a significant effect on the size of your program. The FORTRAN compiler will generate a certain fixed amount of coding for each READ or WRITE:

- READ 3 words
- WRITE 4 words

plus a certain additional amount (average) for each item in the I/O list:

- variable -- e.g., AB or I 2 words
- variable, constant subscript -- e.g., X(3) 4 words
- variable, variable subscript -- e.g., X(J) 5 words
- array name 3 words
- implied DO loop e.g., (X(N), N=1, 6) 19 words

If you wish to WRITE a line containing eight real variables, you may code

\[
\text{WRITE (3, XXX) A, B, C, D, E, F, G, H}
\]

and use \(4 + (8 \times 2)\) or 20 words. Or you could EQUIVALENCE each of the eight items to a variable in the ANSWR array

\[
\text{EQUIVALENCE (A, ANSWR(1))}
\]

\[
\text{EQUIVALENCE (B, ANSWR(2))}
\]

etc.

and code

\[
\text{WRITE (3, XXX) ANSWR}
\]

which would require only \(4 + (1 \times 3)\) or 7 words.

You would not want to use

\[
\text{WRITE (3, XXX) (ANSWR(K), K=1, 8)}
\]

since that would require 23 words, more than the original. In fact, the implied DO loop I/O format should be avoided wherever possible. This can usually be accomplished with the EQUIVALENCE statement. For example, if you want to WRITE the first six items of the eight-item ANSWR array, you would code

\[
\text{DIMENSION ANSWR(8), ANS6(6)}
\]

\[
\text{EQUIVALENCE (ANS6(1), ANSWR(1))}
\]

\[
\text{....}
\]

\[
\text{....}
\]

\[
\text{....}
\]

\[
\text{WRITE (3, XXX) ANS6}
\]

saving 23 - 7 or 16 words.

Avoid Long Subroutine Argument Lists

The coding generated for CALLs to subroutines is quite similar to that of READs and WRITEs -- an initial CALL (two words) plus a certain number of words for each argument:

<table>
<thead>
<tr>
<th>Type of Argument</th>
<th>Approximate Core Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0 words</td>
</tr>
<tr>
<td>Constant -- e.g., 6</td>
<td>1 word</td>
</tr>
<tr>
<td>Unsubscribed Variable -- e.g., X or I</td>
<td>1 word</td>
</tr>
<tr>
<td>Array Name, -- e.g., LARRY</td>
<td>1 word</td>
</tr>
<tr>
<td>Variable with Constant Subscript -- e.g., A(7)</td>
<td>8 words</td>
</tr>
<tr>
<td>Variable with Variable Subscript -- e.g., A(N)</td>
<td>13 words</td>
</tr>
</tbody>
</table>

You can see that there is quite a difference between

a. CALL SUB 2 words
b. CALL SUB (X, Y, Z) 5 words
c. CALL SUB (LARRY) 3 words
d. CALL SUB (A(1), A(2), A(3)) 26 words
e. CALL SUB (A(I), A(J), A(K)) 41 words

There are many ways to avoid those types of CALLs that consume core storage.

Item d, CALL SUB (A(1), A(2), A(3)), could be replaced by

\[
\text{EQUIVALENCE (A(1), X)}
\]

\[
\text{EQUIVALENCE (A(2), Y)}
\]

\[
\text{EQUIVALENCE (A(3), Z)}
\]

and

\[
\text{CALL SUB (X, Y, Z)}
\]

or by

\[
\text{DIMENSION XA(3)}
\]

\[
\text{EQUIVALENCE (XA(1), A(1))}
\]

and

\[
\text{CALL SUB (XA)}
\]

or by placing the A array in COMMON and using CALL SUB with no arguments.

Item e, CALL SUB (A(I), A(J), A(K)), could be replaced by

\[
\text{CALL SUB (I, J, K)}
\]

which would require a revised subroutine but would save 41 - 6 or 35 words. Or it could be replaced by CALL SUB (I, J, K) with the A array placed in COMMON.
Avoid Arithmetic with Variables Having Constant Subscripts

In the average arithmetic statement, a variable with a constant subscript (TOTAL(10)) will require two words more coding than an unsubscripted variable (TOTDF). Such usage can always be avoided by an EQUIVALENCE statement such as

```
EQUIVALENCE (TOTDF, TOTAL(10))
```

Then, rather than say

```
TOTAL(10) = TOTAL(10) + AMT
```
you would code

```
TOTDF = TOTDF + AMT
```

and save two words.

In a large program, the saving can be considerable. Furthermore, it makes the program more readable, since TOTDF can be a more descriptive name than TOTAL(10).

The data can be referred to by either name:
- TOTDF when doing arithmetic
- TOTAL(10) when you want it subscripted -- for example, when clearing an array of totals, when writing an array of totals on the disk, etc.
Reducing Subroutine Requirements

Raising a Real Number to a Whole Power

FORTRAN allows you two ways to do this. For example, to square X, a real number, either X**2 or X*X may be used. While the two look almost identical, the first will use the "real base to integer exponent" routines (about 82 words) and the second will use the "real base to a real exponent" routines (about 242 words).

In this case you should code X**2 and save about 160 words of core storage, unless, of course, your program requires a real base to a real exponent somewhere else.

A programmer will often use this form of arithmetic to obtain the various powers of ten -- for example:

\[ 10^{N} \]
\[ 10^{0} = 1 \]
\[ 10^{1} = 10 \]
\[ 10^{2} = 100 \]

However, if this is the only way in which the double asterisk is used in a particular program, it will usually be more economical to code:

\[
\text{DATA TEN/1.,10.,100.,1000.,etc.} / 
\]

and then use subscripting

...TEN (N+1)......

This will eliminate the 82-word subroutine.

SQRT vs **.5

To take the square root of a number, you have two alternatives: the SQRT function or the 1/2 power option (**.5). While both will give the same result, the core storage required is quite different. The SQRT routine is about 76 words in length; the "real base to real exponent" routine, about 242 words. The difference, about 166 words, is substantial.

Of course, if your program must use the "real base to real exponent" routine (for example X**A), you need those routines anyway. If that is so, use the **.5 option rather than SQRT; otherwise, you will have both packages in core storage.
Don't Include Unneeded I/O Devices on *IOCS Card

In many installations, a stack of all-purpose *IOCS cards is left on the card reader, or nearby, to save the trouble of punching a new card for every program. However, you should be aware that the card *IOCS(CARD, DISK, TYPEWRITER, KEYBOARD, 1132 PRINTER) will cause all those I/O routines to be added to your program, whether you use the devices or not. The size of the package to handle those devices listed above is about 620 words for the disk, and 1780 words for the non-disk group. Because of the way in which the SOCAL system operates, your program may still fit in core, but with more overlays, thus causing it to run more slowly.

It would be wiser to maintain a set of cards with only one device per card

*IOCS(CARD)
*IOCS(1132 PRINTER)
*IOCS(DISK)
etc.

and use only those that are really needed. In this way no unnecessary I/O packages will be included with your program.

Remove FIND Statements If You Have SOCALs or LOCALs

Even if you have included FIND statements in your program, they will not be executed if SOCALs or LOCALs are being used. The FIND subroutine (SDFND), however, remains in core storage.

Therefore, if you know you are going to have SOCALs or LOCALs, remove all FIND statements, and you will save about 80 words of core storage, plus three words for each statement.
Remove the TRACE from Production-Status Programs

The trace features furnished in 1130 FORTRAN are an invaluable aid in debugging. Most users, when they compile their programs, include the *ARITHMETIC TRACE and *TRANSFER TRACE cards, just in case something goes wrong. However, since these features consume both core space and time, they should be eliminated when no longer needed.

Core requirements are increased by about 140 words, and execution time is slowed down for each equal (=) sign, IF statement, or computed GO TO executed. This is true regardless of the status of Sense Switch 15. In addition, the object coding generated may be slightly greater.
FORTRAN EXECUTION TIMES

Processing

It is possible to estimate the length of time it will take to execute an arithmetic block of FORTRAN coding. Inspect your coding sheets, or program listings, and count the average number of times the operations shown in Figure 70.33 will be executed. Then use the times shown in Figure 70.33 to estimate the total execution time.

Note that you must consider the probability of execution, not just the number of appearances. If a certain loop will be executed 15 times, on the

<table>
<thead>
<tr>
<th>Operation</th>
<th>Approximate* time in Microseconds,** each execution (time for standard precision use in parentheses)</th>
<th>Operation</th>
<th>Approximate* time in Microseconds,** each execution (time for standard precision use in parentheses)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>2250 + 2190 C</td>
<td>real =</td>
<td>300 (360)</td>
</tr>
<tr>
<td>PUT</td>
<td>3450 + 3000 C</td>
<td>integer =</td>
<td>22</td>
</tr>
<tr>
<td>EDIT</td>
<td>630 + 90 S + 180 M</td>
<td>+real =</td>
<td>440 (460)</td>
</tr>
<tr>
<td>MOVE</td>
<td>300 + 45 C</td>
<td>+integer =</td>
<td>12</td>
</tr>
<tr>
<td>FILL</td>
<td>300 + 30 C</td>
<td>-real =</td>
<td>490 (560)</td>
</tr>
<tr>
<td>WHOLE</td>
<td>1400</td>
<td>-integer =</td>
<td>12</td>
</tr>
<tr>
<td>NCOMP</td>
<td>250 + 75 C</td>
<td>*real =</td>
<td>790 (560)</td>
</tr>
<tr>
<td>NZONE</td>
<td>350</td>
<td>*integer =</td>
<td>30</td>
</tr>
<tr>
<td>ICOMP</td>
<td>500 + 95 C</td>
<td>/real =</td>
<td>2100 (800)</td>
</tr>
<tr>
<td>NSIGN</td>
<td>240</td>
<td>/integer =</td>
<td>80</td>
</tr>
<tr>
<td>ADD</td>
<td>2160 + 216 L</td>
<td>real**real=</td>
<td>13300 (8000)</td>
</tr>
<tr>
<td>SUB</td>
<td>2160 + 216 L</td>
<td>integer**integer =</td>
<td>4700 (3800)</td>
</tr>
<tr>
<td>MPY</td>
<td>2400 + 120 P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIV</td>
<td>4000 + 0(1445 + 657 DIV)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1DEC</td>
<td>700 + 54 A</td>
<td>FLOAT</td>
<td>330</td>
</tr>
<tr>
<td>DECA1</td>
<td>180 + 117 A</td>
<td>FIX</td>
<td>140</td>
</tr>
<tr>
<td>A1A3</td>
<td>470 + 1084 A</td>
<td>subscript (no variable) =</td>
<td>25</td>
</tr>
<tr>
<td>A3A1</td>
<td>545 + 156 A</td>
<td>subscript (one variable) =</td>
<td>280</td>
</tr>
<tr>
<td>PACK</td>
<td>380 + 63 A</td>
<td>subscript (two variables) =</td>
<td>350</td>
</tr>
<tr>
<td>UNPAC</td>
<td>420 + 66 A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPACK</td>
<td>392 D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUNPK</td>
<td>360 D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIN</td>
<td>5400 (3000)</td>
<td>subscript (three variables) =</td>
<td>530</td>
</tr>
<tr>
<td>COS</td>
<td>5000 (3400)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATAN</td>
<td>8900 (5300)</td>
<td>DO</td>
<td>22 + 50 N</td>
</tr>
<tr>
<td>SQRT</td>
<td>10400 (4500)</td>
<td>IF (real)</td>
<td>190 (210)</td>
</tr>
<tr>
<td>EXP</td>
<td>4400 (2000)</td>
<td>IF (integer)</td>
<td>30</td>
</tr>
<tr>
<td>ALOG</td>
<td>8000 (5100)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TANH</td>
<td>8100 (4300)</td>
<td>GO TO</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GO TO (1, N)</td>
<td>7</td>
</tr>
</tbody>
</table>

N = The number of times through the DO loop
C = Length of the field, in characters
S = Length of the source field
M = Length of the edit mask
P = Length of the multiplier field x length of the multiplicand field (significant digits only — don’t count leading zeros)
A = Length of the A1 field
D = Length of the packed decimal (D4) field
L = Length of the longer of the two fields (significant digits only — don’t count leading zeros)
Q = Number of significant digits in the quotient (result) field
DIV = Number of significant digits in the divisor (denominator) field

* Most timings are approximate and are based on test runs of “typical” cases, using fields of “average” size, magnitude, etc. Unusual cases may or may not differ significantly from the timings obtained from the given equations. This is particularly true of the decimal arithmetic routines (ADD, SUB, MPY, DIV).

** Based on 3.6-microsecond CPU cycle speed. Multiply by 0.6 to obtain timings on 2.2-microsecond CPU.
average, every operation within it should be counted 15 times. If, in the other hand, a certain routine is only executed half the time, it should be counted as half an execution. To illustrate:

```
X=X+6
IF(X-77.)1,2,1
  1  Z=X*14.
  2  Z=X*16./W
  3  CONTINUE
```

If you assume extended precision, and a probability of one-third for path 1 and two-thirds for path 2, the estimated execution time is

<table>
<thead>
<tr>
<th>Operation</th>
<th>No. of Times</th>
<th>x</th>
<th>Unit Time</th>
<th>Total*</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>1+1/3+2/3=2</td>
<td>330</td>
<td>660</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>1</td>
<td>440</td>
<td>440</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>490</td>
<td>490</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>1/3+2/3=1</td>
<td>790</td>
<td>790</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>2/3</td>
<td>2100</td>
<td>1400</td>
<td></td>
</tr>
<tr>
<td>FLOAT</td>
<td>1</td>
<td>330</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>(6 to 6.0)</td>
<td></td>
<td>4307</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF (real)</td>
<td>1</td>
<td>190</td>
<td>190</td>
<td></td>
</tr>
<tr>
<td>GO TO</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

*In microseconds

On the average, then, this portion of your program will require 4307 microseconds, or 4.307 milliseconds, or .004307 seconds.

Figures 70.34 through 70.40 show some additional examples.
### FORTRAN TIMING ESTIMATE WORKSHEET

#### CODING

\[ X = X + G \]
\[ \text{IF } (X - 77.) 1, 2, 1 \]
\[ 1 \text{ Z = } X \times 14. \text{ ONE OUT OF EVERY 3 TIMES} \]
\[ \text{GO TO 3} \]
\[ 2 \text{ Z = } X \times 16.1W \text{ TWO OUT OF THREE TIMES} \]
\[ 3 \text{ CONTINUE} \]

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of Executions</th>
<th>Time per Execution, Microseconds</th>
<th>Extension, Microseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>real</td>
<td>[1+\frac{1}{2}+\frac{2}{3}]</td>
<td>330</td>
<td>660</td>
</tr>
<tr>
<td>+ real</td>
<td>1</td>
<td>440</td>
<td>440</td>
</tr>
<tr>
<td>FLOAT</td>
<td>1</td>
<td>330</td>
<td>330</td>
</tr>
<tr>
<td>- real</td>
<td>1</td>
<td>490</td>
<td>490</td>
</tr>
<tr>
<td>IF(real)</td>
<td>1</td>
<td>190</td>
<td>190</td>
</tr>
<tr>
<td>GO TO</td>
<td>1</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>* real</td>
<td>[\frac{1}{2}+\frac{2}{3}]</td>
<td>790</td>
<td>790</td>
</tr>
<tr>
<td>/ real</td>
<td>[\frac{1}{3}]</td>
<td>2100</td>
<td>1400</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TOTAL =</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

Figure 70, 34.
### FORTRAN TIMING ESTIMATE WORKSHEET

#### CODING

\[
X(i) = x(i) + 6 \\
\text{IF}(x(i) - 77.) 1, 2, 3 \\
1 \ Z = x(i) * 14. \\
\text{GO TO 3} \\
2 \ Z = x(i) * 16./W \\
3 \ \text{CONTINUE}
\]

*Same as Figure 70.34 except that X is subscripted*

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of Executions</th>
<th>Time per Execution, Microseconds</th>
<th>Extension, Microseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>same as Figure 70.34</td>
<td></td>
<td>3 3</td>
<td>4 2 1 7</td>
</tr>
<tr>
<td>Subscripts, I to W</td>
<td>4</td>
<td>280</td>
<td>1 1 2 0</td>
</tr>
</tbody>
</table>

**TOTAL** = 5 3 3 7

---

*Figure 70.35.*
FORTRAN TIMING ESTIMATE WORKSHEET

CODING

`C COUNT TO 1000 - DO LOOP
DO 17 I = 1, 1000
   ..
   ..
17 CONTINUE`

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of Executions</th>
<th>Time per Execution, Microseconds</th>
<th>Extension, Microseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DO loop</strong></td>
<td>1000</td>
<td>22.50 N</td>
<td>50.022</td>
</tr>
</tbody>
</table>

TOTAL = 50.022

Figure 70.36.
<table>
<thead>
<tr>
<th>Component</th>
<th>Number of Executions</th>
<th>Time per Execution, Microseconds</th>
<th>Extension, Microseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer =</td>
<td>1000</td>
<td>22</td>
<td>22000</td>
</tr>
<tr>
<td>-integer</td>
<td>1001</td>
<td>12</td>
<td>12012</td>
</tr>
<tr>
<td>IF(integer)</td>
<td>1001</td>
<td>30</td>
<td>30030</td>
</tr>
<tr>
<td>GO TO</td>
<td>1000</td>
<td>7</td>
<td>7000</td>
</tr>
<tr>
<td>+integer</td>
<td>1000</td>
<td>12</td>
<td>12000</td>
</tr>
</tbody>
</table>

TOTAL = 83042
### FORTRAN TIMING ESTIMATE WORKSHEET

**CODING**

- `C COUNT TO 1000`
- `C STANDARD PRECISION`
- `X = 0.0`
- `1 IF (X-1000) 1,2,7`
- `2 X = X + 1`
- `3 GO TO 7`
- `2 CONTINUE`

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of Executions</th>
<th>Time per Execution, Microseconds</th>
<th>Extension, Microseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>real</code></td>
<td>1000</td>
<td>360</td>
<td>3600000</td>
</tr>
<tr>
<td><code>-real</code></td>
<td>1001</td>
<td>560</td>
<td>560560</td>
</tr>
<tr>
<td><code>IF(real)</code></td>
<td>1001</td>
<td>190</td>
<td>190190</td>
</tr>
<tr>
<td><code>GO TO</code></td>
<td>1000</td>
<td>7</td>
<td>7000</td>
</tr>
<tr>
<td><code>+real</code></td>
<td>1000</td>
<td>460</td>
<td>460000</td>
</tr>
</tbody>
</table>

**TOTAL =** 1,577,750

Figure 70, 38.
**FORTRAN TIMING ESTIMATE WORKSHEET**

**CODING**

- COUNT TO 1000
- EXTENDED PRECISION
  
  \[
  x = 0.0
  \]
  
  \[
  7 \text{ IF}\ (x-1000.)\ 1, 2, 2
  \]
  
  \[
  1\ x = x + 1.
  \]
  
  \[
  60\ TO\ 7
  \]
  
  \[
  2\ CONTINUE
  \]

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of Executions</th>
<th>Time per Execution, Microseconds</th>
<th>Extension, Microseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>real =</td>
<td>1000</td>
<td>330</td>
<td>3300000</td>
</tr>
<tr>
<td>-real</td>
<td>1001</td>
<td>490</td>
<td>490490</td>
</tr>
<tr>
<td>IF(real)</td>
<td>1001</td>
<td>190</td>
<td>190190</td>
</tr>
<tr>
<td>GO TO</td>
<td>1000</td>
<td>7</td>
<td>7000</td>
</tr>
<tr>
<td>+ real</td>
<td>1000</td>
<td>440</td>
<td>4400000</td>
</tr>
</tbody>
</table>

**TOTAL =** 1457680
FORTRAN TIMING ESTIMATE WORKSHEET

CODING

C COUNT TO 1000, DECIMAL ARITH
C ASSUME IX FIELD TEN DIGITS LONG
C ASSUME TWO-DIGIT CONSTANT OF ONE (K1)
C ASSUME TEN-DIGIT CONSTANT OF 1000(K1000)
CALL FILL (IX, 1, 10, 0)
7 IF (NCOMP (IX, 1, 10, K1000, 1, 10)) 1, 2, 2
1 CALL ADD (IX, 1, 10, K1, 1, 2, NE)
   GO TO 7
2 CONTINUE

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of Executions</th>
<th>Time per Execution, Microseconds</th>
<th>Extension, Microseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF(integer)</td>
<td>1001</td>
<td>30</td>
<td>300</td>
</tr>
<tr>
<td>GO TO</td>
<td>1000</td>
<td>7</td>
<td>7000</td>
</tr>
<tr>
<td>FILL</td>
<td>1</td>
<td>300+10×30</td>
<td>600</td>
</tr>
<tr>
<td>NCOMP</td>
<td>1001</td>
<td>250+10×75</td>
<td>100</td>
</tr>
<tr>
<td>ADO</td>
<td>1000</td>
<td>2160+4×216</td>
<td>3024000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL =</td>
<td></td>
<td></td>
<td>4062630</td>
</tr>
</tbody>
</table>

Figure 70, 40.
Summary and Conclusion

From the examples shown you may draw some conclusions:

1. Integer arithmetic is much faster than real arithmetic.
2. Extended precision and standard precision real arithmetic are of essentially the same speed.
3. Decimal arithmetic is fairly slow.
4. Subscripting adds a considerable amount of time to arithmetic calculations. (It also increases the size of your program.)
5. Unnecessary use of mixed mode expressions can add somewhat to execution time.
## FORTRAN TIMING ESTIMATE WORKSHEET

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of Executions</th>
<th>Time per Execution, Microseconds</th>
<th>Extension, Microseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TOTAL =**
Section 75: SORTING WITH YOUR 1130

CONTENTS

Introduction ................. 75.01.00
Some Preliminary Information ....... 75.10.00
Alternate Approaches ............ 75.20.00
  Use of File Organization ........ 75.20.10
    Pure Sequential
    Indexed Sequential
    Random
Sorting Offline ............... 75.20.20
Methods of Sorting ............. 75.30.00
  Introduction ................. 75.30.01
    Key Compare vs Key Value
      (Radix) Techniques
    Sequence-Creating vs
    Sequence Reducing Techniques

Degree of Data Accessibility
  Degree of Generality
Internal Sorting Methods .......... 75.30.10
  Selection
  Exchanging
  Merging
  Insertion
  Replacement Selection
  Address Calculation
External Sorting Methods .......... 75.30.20
  Key (Tag) Sorting
  Key Sort vs Record Sort
A Detailed Look at an 1130 Record
Sort ................................ 75.40.00
Summary ........................... 75.50.00
INTRODUCTION

Most data processing applications require a sequential arrangement of the information to be processed. Frequently, a collection of related information, or file of data records, is to be updated by adding, deleting, or changing information as new transactions occur. Before the new transactions can be applied against the main or master file, however, a method must be established whereby a transaction can be associated with a master. One such method would be to arrange the transactions in the same sequence as the master file (see Figure 75.1). For this purpose, the master and transaction files are sequenced by some common identifying characteristic, such as part number, account number, employee number, etc. Similarly, when payroll earnings are to be computed or data is to be tabulated in accordance with some scheme of classification, it is necessary to arrange the information in a sequence that facilitates processing.

Sorting is simply a systematic method for arranging or rearranging a file of data records in sequence by some group of characters that constitute the control field, or control word, of the record. (Control words are sometimes called the key.)

This section discusses sorting with your 1130 but attempts first to show you (1) a possible way to avoid sorting with your 1130 and (2) a way to ease the task of writing a sort program, if one must be written.

Figure 75.1. Transaction file and master file in same sequence
SOME PRELIMINARY INFORMATION

It may be useful to review the meaning of some basic terms and concepts that are part of sorting terminology. As already stated, sorting concerns the arrangement of a file which is a collection of related data records stored in a data storage medium (cards or disk). The file size specifies the total number of records contained in the file. The input file is the collection of data records introduced as input to the sorting process, while the output file represents the collection of records properly sorted and stored.

To place a file into a specified sequence, each of its records must somehow be uniquely identifiable. The identification is made by means of the control key, a group of characters arranged in a certain way. The contiguous groups of characters that are placed in order within the control key are called control fields. Each of the control fields bears certain identifying information, such as payroll number, name, organization code, address to which checks are sent, etc. The data record control field that is most important in sequencing the records is called the major control field. When two records contain identical data in their major control field, they must be compared by the next most significant, or minor, control field in order to be sorted into the proper sequence. If even the minor control fields are equal, the next most significant or minor control field must be considered, and so on. Thus, for the purpose of successive comparison, all the control fields within the control key are arranged in major-minor (that is, decreasing) order of significance (see Figure 75.2).

Since the control fields of a record may consist of numbers, letters, or special characters ($, -, +, etc.), an order must be prescribed for the characters of the control field to determine which is greater and which is less. Such an order of characters, upon which the sequencing of records is based, is known as the collating sequence. In the 1130, the collating sequence is A-Z, 0-9, blank, and special characters, in ascending order (see 70.40.20). The collating sequence determines the proper order of the control keys.

Using these definitions, sorting may now be defined more accurately as the process whereby a file of records is placed in order by the collating sequence of the control keys of the records.

A considerable body of specific sorting terms has been generated over the years. To simplify the ensuing discussion, some of the more commonly used terms are explained here.

The object of a sort is (to restate it) to place a file of records in a desired sequence. Any group of data records in which the control keys are in the desired collating sequence is called a "sequence" -- or, sometimes, a "string". The length of each sequence can be one or more data records. It has been assumed till now that a sort must be in ascending sequence; that is, the final sequence of records is such that the control key of each successive record collates (compares) equal to or higher than that of the preceding record. This need
not be the case, however. A sort can be in a descending sequence, with the control key of each successive record collating equal to or lower than that of the preceding record.

Frequently, two or more sorted files have to be merged into a single file of sequenced records. In general, "merging" is a technique that collates several sequences of data records to form a single sequence. The number of files to be combined during a merging operation is known as the order of merge, or "merge order". Thus, a merge of order m is called an "m-way merge". The processing of all the records once through the merge is termed a "merge pass", or simply, a "pass". The object of a pass is to reduce the number of sequences (strings) by increasing the number of records contained in each sequence. During a single pass, the number of sequences is usually reduced by a factor equal to the order of merge (m). Several intermediate passes may be required to reduce the file to a single sequence. A multi-pass sort is a sort program designed to sort more data than can be contained within the internal storage of the central processing unit. In this case, intermediate storage (disk) is required.

It is customary to segment a sort program into a number of phases, each of which is executed as one core storage load. For example, a typical sort may be divided into four phases: an initialization phase, an internal (presort or premerge) phase within core storage, a merge phase (for combining the sequences), and a final output phase.

The sequencing of a group of data records contained at one time in core storage is known as an "internal sort". The size of the internal sort is the number of data records (abbreviated G) that can be sequenced at one time in core storage. Note, however, that since the number of data records to be sorted usually exceeds G (the number contained at one time in core storage), the internal sort process must generally be repeated until all the records in the file have been sequenced into strings that may later be combined, or merged.

It has been implied that sorting consists of moving data records around until their respective control keys are in the proper collating sequence. This is not always the case. In some sorting methods, the control keys upon which sequencing is based are read from the record and combined with the record number (called tag) to form a key-tag pair. Then the keys are sorted, rather than the original records. After sorting, the tags serve as an index for later retrieval of the data records in the desired sequence (see Figure 75.3).

The effectiveness of a sort program is measured by the time it takes to sort a file of data records. If the sorting method alone determined the overall performance and speed, the choice of the best method would be relatively simple. In actuality, though, sort performance is the result of a complex interaction between the characteristics of the data file, the data processing system, the sorting method used, the objectives desired, and a number
of other characteristics. Thus a great many factors play a role in determining the efficiency and speed of a sort program.

Among the more important data file characteristics, the following may be cited: the degree of original ordering of the file (that is, is it in random order or do natural sequences exist?); the length, range, and location of control word data; and the number and length of the records.

Equally important in influencing sort performance are the characteristics of the storage facilities and the CPU of the computer. Among storage characteristics of interest are the capacity of the main internal storage and the mode of addressing it, as well as the availability and access times of external storage devices, such as disk files. Relevant machine and CPU characteristics include simultaneous read, write, and processing capability; the basic processing speeds of compare, add, and move operations; the structure of the OP-code set; and the availability of indexing, table lookup, etc.

For a given sorting method, the data file characteristics influence the primary sorting statistics, such as the total number of arithmetic operations or comparisons and the total number of passes. For a file of a given size, each method also has some inherent characteristics that influence the complexity and speed of the sort -- for example, the required working storage, the required number of comparisons, transfers, and exchanges, etc.

Finally, realistic sorting objectives must consider the specific data processing requirements, as well as the complexity and cost of the sort programming effort. A specific sort program should try to provide an optimum match between the specified data file, the given machine configuration, and the chosen technique. In sorting large files, a single sorting technique cannot always provide this optimum match. Frequently, therefore, a program combines two methods in order to take advantage of special machine features, minimize the effects of storage limitations, and provide increased speed.
ALTERNATE APPROACHES

Before you write a sort program for your 1130, examine your files and the reports to be produced from them. You may find that sorting on your 1130 is not necessary, or that sorting can be avoided.

Some alternate approaches to sorting on your 1130 are:

- Use of file organization
- Sorting offline
Use of File Organization

Is it possible to keep multiple copies of your files, each in the sequence of a report to be produced? If so, you can avoid sorting. If not, however, as is likely with moderate to large files, the importance of your file organization scheme emerges.

Pure Sequential

An answer for files organized in a pure sequential manner is to maintain multiple copies on multiple disk cartridges. This eliminates sorting but may cause problems in processing. (See Figure 75.4.) Generally, with pure sequential files that are too large for multiple copies, the solution is offline sorting.

Indexed Sequential

Is it possible to keep multiple copies of your index, with each index in the sequence of a report to be produced? Since your index is considerably smaller than your file, this may be the ideal solution. Processing against the file would be random. (See Figure 75.5.) Again, if this solution cannot be used, you can still sort offline.

Random

In this case your files are usually organized in a sequence that does not relate to a report. The transactions (say, cards containing only control keys) must be sequenced appropriately; a sort is necessary. Hence, the only way to avoid sorting using your 1130 is to sort offline.

Figure 75.4. Same data in two files, but in different sequence
Figure 75.5. One file, but with a multiple index system
Sorting Offline

Sorting offline can be either a manual or a mechanized procedure. A manual procedure (by hand) should not be used unless volumes are very small. Even with small volumes, you will need a program to sequence-check the sorted cards.

A mechanized procedure involves the use of a sorter. IBM has mechanical sorters available that can process up to 2000 cards per minute.

The rule-of-thumb procedure for timing offline mechanized sorts is:

1. Compute the card-passing time for each column in the control key.
2. Sum these times.
3. Add 10% for card handling.

You must decide whether the time and money spent sorting offline will be less than the cost of programming and running a sort for your 1130.
METHODS OF SORTING

Introduction

Sorting and merging methods can be classified in accordance with certain distinguishing characteristics.

Key Compare vs Key Value (Radix) Techniques

Most sorting methods compare control keys of two or more records at a time and sequence the records on the basis of a high, low, or equal comparison of the keys. Despite variations, all key compare techniques are essentially similar in concept. An example of a key compare technique is the card player's way of inserting new cards into his hand in proper sequence, by comparing the value of each new card with the values of those he is already holding.

In some sorts, action is taken on the basis of the value of the individual digits in the key and their position, rather than by comparison of two keys. The value of the key digits -- or, more generally, of the key number base (radix) -- is used to determine into which particular slot each record should go. Key value or radix techniques are also known as digit sorts, which is a narrower term. The mechanical punched card sorter, with separate pockets for each key value, is an excellent example of a radix technique. Another illustration is the distribution of a deck of cards into four piles (or files), one for each suit.

Sequence-Creating (Internal) vs Sequence-Reducing (External) Techniques

Another fundamental way of viewing sorting is to distinguish between techniques that create sequences (starting with a random or unsorted file) and those that reduce the number of existing sequences to one. In theory, most techniques capable of creating sequences of at least two records, or keys, are also capable of lengthening those sequences to a point where, finally, all records are contained within a single sequence. In practice, however, the sequence-creating or internal sorts are usually only the prelude to the main or merge phase of the sort (hence the terms "presort" and "premerge"). Initial sequences are created by loading a group of records into core storage, sorting the records internally, and placing the resulting sequence on an intermediate storage device. This internal sort process is repeated until the input file is exhausted. The sequences thus created internally are then reduced to one by an external merge. If the entire file can be contained within core storage at one time, the sort is exclusively internal. In most cases, however, both internal (sequence-creating) and external (sequence-reducing) techniques are necessary to sort a large file.

Degree of Data Accessibility

Sorts also may be distinguished in accordance with their relative need of data accessibility. Most of the internal techniques are best suited to storage media that can provide rapid access to many groups or sequences of records. Core storage provides the most rapid and direct access, while disks furnish a lesser degree of data accessibility. A number of methods work well with disk.

Degree of Generality

Finally, sort programs may be categorized by the relative degree of specificity or generality for which they are designed. A large range of objectives exist between narrow, highly specific sorts and broad, generalized programs. On one end of this range there are specific sorts designed to operate on a specified input file and a specific computer configuration. Somewhere in between are generalized sorts that will accept the introduction of some parameters at execution time to adapt the sort program to the characteristics of the particular file and computer configuration. At the other extreme of the range, there are highly sophisticated, generalized sorts and sort generators that, virtually without user intervention, can generate a great variety of ordered results on a variety of file and computer configurations.

In most instances, a specific sort program will satisfy your sorting needs. The remainder of this subsection discusses some sorting methods (both internal and external) of the types described above. In addition, one of the easily implemented sorts is expanded in flowchart form for your more detailed examination.
Internal Sorting Methods

Internal sorting is defined as the sequencing of a group of data records contained in the core storage of your 1130. It generally involves reading successive records from disk storage into core storage, sorting the group in storage by one of the methods to be described, and then writing the sequenced group onto disk.

Since internal sorting is generally a part or a phase of other processing and programs, you must distinguish between methods according to the ultimate purpose they serve. Thus, some sorting routines found in compilers, assembly programs, and other applications are strictly internal; that is, a group of items is to be sequenced only in core storage, not written onto disk. On the other hand, in most generalized and specific sort programs, the file of records is too large to be contained, at one time, within core storage. Here the internal sort passes serve only as a prelude to the subsequent external merge phase of the sort and, hence, are frequently called presorts or premerge sorts. The purpose of the internal sort, then, is to form a number of sequences, or strings, which are placed into the output and subsequently merged. The more efficient the premerge sort, and the longer the strings it generates, the fewer external merge passes required.

In addition to the purpose of the sort, the following considerations apply in selecting an internal sort technique and evaluating its suitability for a specific application:

1. Characteristics of the machine (basic processing speed, internal storage capacity, etc.)
2. Input/output characteristics (number of disk drives).
3. Number and length of data records.
4. Length and range of control keys.
5. Degree of original file ordering (natural sequences).
6. The associated program.

Since there is no single best method for all types of applications, most sort programs represent a compromise between conflicting requirements. In general, they attempt to incorporate the following in as nearly optimal a manner as possible:

1. Sort internally as many records as can be packed into core storage.
2. Minimize total process time per record.
3. Function in a manner compatible with I/O operations and strive for a maximum overlap of read, write, and processing time.
4. Utilize existing sequences in the input file, if possible.
5. Write routines that are compact and that can be modified easily.
6. In a generalized program, accept and sort variable length records with any size control key.

Generally, records can be sorted by (1) physically moving them about until they are in order, (2) forming tables of record numbers (tags) in storage, which are then sorted, or (3) combining the control key and record number and sorting the resulting short key-tag pairs. Either tag sorting or key sorting is the preferred method today. The sorted keys are then used as an index to the file.

In addition to an explanation, the advantages and limitations of each sorting method will now be evaluated briefly with respect to major file and machine characteristics. Additional methods and additional information on each of the methods discussed may be found in Sorting Techniques (C20-1639).
Selection

Sorting by selection -- perhaps the simplest, and also the slowest, of the internal sorting methods -- consists essentially of an examination of the input file to find the record with the smallest key (for an ascending sort) and placing this record or its key in the output area as the first item of the new file. The source file is then scanned for the smallest key of the remaining records, which becomes the second item of the new file, and so on, until all items have been placed in the output file.

When the selection process is carried through the entire file in one stage, it is called "linear selection"; when the original file is broken up into groups, and the smallest key of each group is chosen, and then the smallest of these smallest keys, the process is termed "quadratic selection".

Selection requires a relatively small working storage area in core, equal to the number of items being sorted internally. However, the number of passes over the file also equals the number of items (one for each record), and the total number of comparisons required increases with the square of the number of items to be sorted (for linear selection); this rapidly becomes inefficient for a large file.

Exchanging

The technique of sorting by exchanging consists essentially of comparing the keys of successive records -- either one by one or pair by pair -- and exchanging out-of-sequence items. The sort is completed when no exchanges are made during a pass through the file. Many variations of this general procedure are possible.

The major advantages of exchange techniques are the relative ease of their programming and the fact that all work is done in the area in which the original file is stored; no separate working storage area is required. Among the drawbacks are the dependence of exchange methods upon the distribution of the control fields in the original file and upon the number of records in the file. If the file is almost in sequence, one pass will generally suffice. In the worst case, reverse sequencing, the number of passes may equal the number of items (G) to be sorted, and the number of exchanges (key and/or record movements) may become very large. Since the number of comparisons required increases with the square of the number of items to be sorted, exchange methods are most efficient for sorting a relatively small file of records. Perhaps the simplest exchange technique, and the easiest to program, is pair exchange. The keys of adjacent records are compared; whenever they are not in ascending sequence, they are interchanged. During the first pass, the keys of the first and second records are compared, then of the second and third, of the third and fourth, and so on, until all keys in the file have been compared and interchanged, when necessary. Each successive pass will process one less record. The sort is completed when no interchanges occur during a pass. The example below illustrates the procedure. In general, the maximum number of passes (for the worst case) is equal to G - 1. The average total number of comparisons (C) is

\[
\frac{G(G-1)}{2}
\]

where G is the total number of items to be sorted.
The size of the file is of great importance, since the total number of comparisons and interchanges increases roughly with the square of the number of records in the file.

Merging

Merging is the process of combining several sequences of records to form a single specified sequence. The same rules by which sequences are combined may also be used to form sequences (of two or more items). Thus, the merging process has, essentially, a dual nature: it can be used for creating sequences (usually in an internal sort), and it is also capable of reducing previously created sequences to one (usually in an external sort). This dual capability contrasts with the selection and exchange techniques described thus far, which are useful primarily for internal sorting of relatively short files of records. The versatility, speed, and simplicity of merging make it one of the most widely used sorting techniques.

There are two basic methods of merge sorting: (1) straight or standard merging, with fixed-length sequences, and (2) natural merging, with variable-length sequences, or strings. (The words "sequence" and "string" are often used interchangeably in merging terminology.)

In straight merging, the input file is distributed initially into two or more work areas, depending upon the number of sequences to be combined during each merge (that is, the order of merge). For example, in a method of two-way straight merging, the first merge pass alternates between two storage areas to form strings of two records, one from each area. Subsequent passes double the length of the strings each time (for example, 4, 8, 16, etc.), until the last pass produces a single sequence of all the records. The length of the strings during each pass and the number of passes are fixed.

The natural merge sort takes advantage of "natural" sequences in the original file, which occur with a certain "probable" frequency. The length of the strings on each pass is no longer fixed, but depends upon the existing sequences. The total number of passes required to sort a given file, then, also depends on the number of natural sequences in the original file. For a file that is in correct sequence, only a single pass is required -- to verify that sequence. In the worst case, the number of passes is the same as for straight merging.
Insertion

A fairly effective method for sorting a small number of items, the insertion technique, places each item in sequence as soon as it is encountered. The records (or tags) are brought into storage one at a time, the key is examined, and the item inserted in the correct place of an output file. Earlier members of the partial file are moved aside, when necessary, to make room for new items. The method is straightforward and easy to program, but is relatively slow compared with other techniques.

Sorting by simple insertion has two inherent drawbacks: (1) the partial file must be searched each time to locate the correct place for inserting the new item, and (2) excessive shifting of the sorted records is necessary for each new insertion. The first limitation can be overcome, to some extent, by subdividing the area that must be searched in order to locate the correct position of each new item. The second drawback -- the large amount of record movement -- can be avoided by sorting record numbers (tags), rather than the record themselves. Even with these improvements, the method is too slow for larger files.

Replacement Selection

The internal sorting methods described thus far are all capable of sorting a group of records \((G)\) that can be contained at one time in core storage. The maximum string length is, therefore, limited to \(G\) items. An auxiliary technique, known as replacement (sometimes, replenishment), tries to keep the core storage area filled with \(G\) items by replacing records that have been withdrawn during the sort. As a result, for a file in random order, an average string length of \(2G\) items is developed in an area with a capacity of only \(G\) records. For a given amount of available core storage, the replacement technique produces the maximum possible sequence length. This characteristic makes the technique eminently suitable as a premerge sort and permits a significant reduction in the number of merge passes required for a subsequent external (disk) sort. The price paid for this advantage is increased complexity of programming, relatively long processing time per record, and a slight increase in the required working storage. Also, the number and length of the sequences are variable and, hence, not predictable. Most replacement sorts, however, will generate string lengths approximating \(2G\).

Essentially, the replacement selection method determines the lowest record in the record storage area, moves it to the output area, and then replaces it with a new record from the input file. If the new record is lower than the one just moved to the output, it cannot be part of the current sequence and, therefore, is flagged or held for the next sequence. The process then continues with the selection of the next-lowest record, and so on, until there are no more replacement records in the record storage area that fit into the current sequence. A new sequence is then started, and the procedure continues until the entire input file is processed.
Address Calculation

When the approximate distribution of the key values is known, it becomes possible to sort a file internally by estimating the eventual (sorted) position of each key. This method is called "address calculation" or "pigeonhole sorting".

Briefly, it consists of calculating the correct record number of each item within the file by a predetermined linear formula of the form \( y = a + bx \). If the location at that record number is empty, the item (record or key) is placed there; if it is full, a search is made to find the closest empty space in the vicinity of the calculated record number. The item at the calculated record number and the adjacent items are then moved so that the new item can be inserted in its proper place in the sequence.

Address calculation is similar to the insertion method in that each item is placed directly in its proper relative position within the file, and the entire file is in order just after the last item has been inserted. The method differs from insertion, however, in that some foreknowledge of the range and distribution of the keys is required to estimate the relative location for each item. When this is available, address calculation is a relatively simple and rapid method for sorting a medium-size file (several hundred to a few thousand items) of small to medium-length records. The major disadvantage of the method is the need for a fairly large storage area -- about two or three times the size of the area needed for the original file. If only a relatively small working storage area is available, or if the distribution within the file is not as forecast, a great deal of processing time will be spent in redistributing the records.

To illustrate this method, let us consider a hypothetical case: Many years ago, the ABC Company set up a man-number system based on a three-digit number. Since they had about 150 employees, each man was assigned, in alphabetic order, a number evenly divisible by 5 (005, 010, 015, 020, 025, ..., 995). However, there are now about 240 employees, and the system is not quite as neat as it once was.

Some of the men (50 of them) have been assigned numbers out of the normal pattern (for example, 862 in between 860 and 865). They are still in alphabetic order, though.

The address calculation sort could be used to place this employee file onto the disk in alphabetic (man-number) sequence in the following way:

1. Set up a file containing 500 records.
2. As each man-number is encountered, divide it by 2.5, and convert the result to an integer (call it \( N \)).
3. Check record number \( N \) to see whether there is already an employee there.
4. If there isn't, put the man just processed into that record.
5. If there is someone there already, move the adjacent records up (or down) until there is room to insert the new man.

This will be quite fast, provided the "moving around" (step 5) is not required too frequently. If it is, the file could be increased to 600 records, and the man-number divided by 2. This, however, would waste a considerable amount of space on the disk.
External Sorting Methods

When a file cannot be contained within core storage, additional external passes and intermediate storage devices, such as disks, are required to sort the file. The internal sort, then, is only one phase of a generalized multiphase (or multipass) sort that may have three or four phases. In such a multiphase sort, the internal sort phase is concerned with the creation of suitable sequences from the main file, while the external sort, or merge phases, are devoted to the reduction of those sequences to one continuous sequence.

Practically all the internal sorting techniques described earlier can also be used -- with varying success -- for external sorting by changing the terms of reference appropriately. Thus, the internal storage area is replaced by several input and output areas on disk.

It has been suggested earlier that sorting techniques could be categorized according to the degree of and relative need for data accessibility. Thus far, sorting techniques suitable for one extreme of data accessibility have been described. The internal sorts were seen to be best suited to high-speed, direct (random) access storage media, such as magnetic core storage. In these media, any record or string of records can be accessed immediately, without the need for passing over other, unwanted records.

Despite their name, direct (random) access file storage media (such as disks) provide a degree of data accessibility less than core storage. The time to access a record in these devices is not completely independent of the location of the previously accessed record (as in core storage), but neither does it depend on the entire sequence of records stored before it (as in magnetic tape). The time to access the next record depends on the number of cylinders the access mechanism must be moved from the previous record. (Each one or two cylinder move on the 2310 disk drive requires 15 milliseconds.) However, since internal core storage is generally insufficient to hold an entire file, auxiliary storage devices such as disks are usually necessary.

With disks some attention must be given to the relative advantages of key or tag sorting and sorting of complete records. It has been found in internal sorting that key or tag sorting (involving either record numbers only or short control records) is considerably faster than sorting of complete records. However, because of the substantial seek time, this is no longer true for disks, when the original records must be retrieved at the end of the sort.

The following paragraphs explore some of the considerations pertinent to disk sorting.
Key (Tag) Sorting

In general, key sorting consists of extracting the control key from each record and adding the record number to form a key-tag pair. These pairs, rather than the original records, are then sorted. (Sorting is done with the key; the record number is merely moved about so as to remain with its associated key.) After sorting is completed, the pairs provide an index for later retrieval of the data records in proper sequence. The obvious advantage of key sorting is the more rapid processing of the key-tag pairs, rather than the much longer original records. During internal sorting, more pairs can be sorted into strings; thus, fewer strings and, probably, fewer merge passes will result. The eventual retrieval of the data records (if needed) from external storage is done using the final sorted key-tag file.

A typical key sort with disk storage proceeds in either two or three phases, depending upon whether final retrieval of the data records is necessary. Phase 1 is an internal key sort; phase 2 merges the internally formed strings of key records; and phase 3, if required, retrieves the input records in proper sequence. The approximate procedure during each phase is described below.

Phase 1 (Internal Sort) consists of the following steps:
1. Place input records on the disk file in order of occurrence.
2. Form key-tag pairs by lifting the control field(s) from each input record and adding to it (them) the disk record number.
3. Read G key-tag pairs at a time into core storage and sort them internally (by any standard method) into strings of length G. (G refers to the number of items that can be contained at one time in internal core storage.)
4. Write the strings of G pairs successively onto the disk file, using as many sectors or files as necessary (usually no more than five files of strings).

Phase 2 (Merge). The merge phase of the sort consists of merging the strings of pairs from the separate files on disk. The merge is completed when a single sequence of key-tag pairs has been written onto disk. During the final merge pass, the control keys are stripped from the key-tag pairs, leaving only the disk record numbers or tags.

These record numbers then serve as an index for placing the input records in sequence. At your option, sorting can end at this point.

Phase 3 (Record Retrieval). This phase is necessary if the data records are to be retrieved from the disk file in their sorted order. (Remember, only the tags have been sorted, not the records themselves.) The manner in which this is done has a greater effect on overall timing than phases 1 and 2 combined. The simplest way (also the slowest) consists of retrieving the records one by one in the order indicated by the successive disk record numbers. If the original input records constitute a large file extending over several cylinders of the disk, the probability is high that a seek must be executed for the retrieval of each record. This will add considerably to the time required, since the seek time necessary to retrieve the records will probably dominate the overall sort time.

A number of ways have been devised to minimize this seek time during the retrieval of records in phase 3. One method consists of bringing the disk record numbers (from phase 2 of the sort) into internal storage in some multiple of the output blocking factor. The disk record numbers are then sorted internally in ascending sequence, thereby reducing the seek time between records. The input records are read initially from the disk in ascending record number sequence; blocks of records are then placed in proper sequence (in accordance with the original sequences of disk record numbers); and the sorted records are finally written back onto the disk file. The method reduces seek time substantially, at the expense of more complex programming.

Another method of modifying the key sort consists of blocking the sorted keys so that the number of items in each block plus an equal number of original records just fills the core working area. The items in each block are then sorted again to place the disk record numbers in ascending sequence. As before, the records indicated in each block can then be retrieved sequentially from the file and sorted internally into the proper sequence.

It will be found, however, that in most cases, and for large files in particular, these methods of reducing the seek time still result in a greater overall sort time than might have been required to perform a complete record sort.
Key Sort vs Record Sort

Usually, key sorting is of no advantage, even with large disk files, when most or all of the original records are to be retrieved. Modifying the sorting and reading schemes to minimize the total seek time can have a considerable effect, but the advantage, generally, still lies with record sorting. Whether a record sort or a key sort should be used to sort a disk file depends largely on the ultimate disposition of the sorted records.

If only an index of sorted records is necessary, and few of the sorted records are actually used, key sorting would appear to have the edge. Exception reports extracted from the sorted file are an example of this type of situation. On the other hand, if most or all of the original records are to be retrieved, record sorting is preferable to key sorting. Moreover, the advantage increases with the size of the file.
A DETAILED LOOK AT AN 1130 RECORD SORT

An improvement on the simple exchange technique consists of making alternate passes in opposite directions, attempting to move the high records to the bottom and the low records to the top of the file. This is called an alternating pair exchange sort.

The procedure starts in exactly the same manner as in pair exchange, by comparing the keys of successive records. After an exchange is made, the high key is compared with the key of the next record in sequence, and these comparisons continue until either a higher key is found or the end of the file is reached. All intermediate records (and their keys) are shifted up one position. During this first downward pass, therefore, a high record can move down many positions, but lower records can move up only one position.

The second pass is in the upward direction (from the bottom to the top of the file) and tends to move the smaller records closer to the beginning. During this, and during every other even-numbered pass, a high record can move down only one position, but a low record can move up many positions. Successive passes continue to alternate, with odd-numbered passes in ascending sequence and even-numbered passes in descending sequence. The file is in sequence when no interchanges occur during a pass. A final output pass is required to verify the correct sequence.

The example below illustrates the alternating exchange technique. The first pass, proceeding downward, recognizes that 89 and 56 are out of sequence and exchanges them. The high of the compare, 89, is then compared, in turn, with 02, 08, and 21; since 89 is higher in each case, it moves to the bottom of the file. The low of the compare, 56, moves up one, and all intermediate keys also move up one position. In the second pass, the comparisons start with 89 and move upward. The first out-of-sequence keys are 02 and 56. The 56 drops down one position, while the 02 moves up two positions, since it is lower than the 13. During the next downward pass, the 56 and 08 are out of sequence; the 08 moves up one position, and the 56 moves down two positions, since it is higher than the intermediate 21. During the fourth (upward) pass the out-of-sequence 08 and 13 are interchanged. The final output pass is needed to check the completion of the sort.

<table>
<thead>
<tr>
<th>Input</th>
<th>End Pass 1</th>
<th>End Pass 2</th>
<th>End Pass 3</th>
<th>End Pass 4</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>13</td>
<td>02</td>
<td>02</td>
<td>02</td>
<td>02</td>
</tr>
<tr>
<td>89</td>
<td>56</td>
<td>13</td>
<td>13</td>
<td>08</td>
<td>08</td>
</tr>
<tr>
<td>56</td>
<td>02</td>
<td>56</td>
<td>08</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>02</td>
<td>08</td>
<td>08</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>08</td>
<td>21</td>
<td>21</td>
<td>56</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>21</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
</tbody>
</table>

The arrows indicate the direction of the pass. The example shows that the maximum number of passes is equal to the distance, measured in number of keys or records, which is the largest separation of a key from its final place in the sequence. In this case, 89 is four positions away from its final (bottom) position in the file and, therefore, at most four passes (plus the output pass) are required to complete the sort. In general, the alternating exchange method requires slightly more complex programming than the earlier exchange method, but it results in a smaller number of compares and, frequently, fewer passes. The following pages show this method in more detail.
Move bottom of NARAY to top when $M > 0$, top to bottom when $M < 0$

NREC1 = NSTRT

DISK

1M,NREC2,NRL, NRPC,NARAY, NRECT

FIND

NREC1

SORT

NEXS,LARRY, NRL, KS, KE

N = N + NEXS

M = 1	n = 0

NE XS,LARRY, NRL,KS, KE

STOP

5

M = - M

Move bottom of NARAY to top when $M > 0$, top to bottom when $M < 0$

NRECT = NEND

NSTRT = NREC1

2M,NREC 2 = (M)* (NRPC),NRL,NRPC, NARAY, NRECT

DISK

2,-M,NREC1, NRL,NRPC, NARAY, NRECT

STOP

NRECT = NSTRT

NEND = NREC1

NREC2 = NREC1 + (M)NRPC

A

A

A

Section Subsections Page
75 40 00 02
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>INPUT TEMP OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>A subroutine to (1) fill the RSA with unsorted records and (2) empty the RSA of sorted records</td>
</tr>
<tr>
<td>FIND</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>The FORTRAN statement to move the disk access mechanism in an overlapped fashion</td>
</tr>
<tr>
<td>*KE</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>user</td>
<td>option 1</td>
<td>End of the control key, its ending position</td>
</tr>
<tr>
<td>*KE</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>user</td>
<td>option 1</td>
<td>Beginning of the control key, its starting position</td>
</tr>
<tr>
<td>*LARRY</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>user</td>
<td>option 5120</td>
<td>Length of NARAY, the Record Storage Area. Should be at least two cylinders (5120). Make maximum in allowable core.</td>
</tr>
<tr>
<td>M</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>-1</td>
<td></td>
<td>Direction of sort, up the file or down the file, -1 or +1.</td>
</tr>
<tr>
<td>N</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td>A count of the number of exchanges in a pass</td>
</tr>
<tr>
<td>NARAY</td>
<td>A</td>
<td>E350</td>
<td>-</td>
<td></td>
<td></td>
<td>Name of array for Record Storage (RSA).</td>
</tr>
<tr>
<td>*NEND</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>user</td>
<td>option 2</td>
<td>The record number of the last record in the file</td>
</tr>
<tr>
<td>NXES</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>up to</td>
<td>NARAY 0</td>
<td>The number of exchanges during the sort of a pair of cylinders.</td>
</tr>
<tr>
<td>NRECT</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>NEND</td>
<td>NSTART</td>
<td>The record number ending a pass</td>
</tr>
<tr>
<td>NREC1</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>NEND</td>
<td></td>
<td>The record number for writing back to the file</td>
</tr>
<tr>
<td>NREC2</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>NEND</td>
<td></td>
<td>The record number for reading from the file</td>
</tr>
<tr>
<td>*NRL</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>user</td>
<td>320 2</td>
<td>The record length.</td>
</tr>
<tr>
<td>NRPC</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>1280 8</td>
<td></td>
<td>Number of records per cylinder.</td>
</tr>
<tr>
<td>NRW</td>
<td>I</td>
<td>T</td>
<td>T</td>
<td>2</td>
<td></td>
<td>READ/WRITE switch, Read = 1, Write = 2.</td>
</tr>
<tr>
<td>*NSRT</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>user</td>
<td>option 1</td>
<td>The record number of the first record in the file</td>
</tr>
<tr>
<td>SORT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>A subroutine to sort the contents of NARAY, the Record Storage Area.</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic.
\[ L = (1 - 1)NRL + 1 \]
\[ N_{l} = NRL \]
\[ L = 1 + (NRPC + 1 - 1)NRL \]
\[ N_{l} = (NRPC + 1)NRL \]

\[ I = 1 + 1 \]

Read Disk
\[(N_{F}'NREC)\]
\[ NARAY (J) \]
\[ J = L, N \]

Write Disk
\[(N_{F}'NREC)\]
\[ NARAY (J) \]
\[ J = L, N \]

\[ NREC = NREC + M \]
\[ NREC = NREC - M \]

\[ (M)(NRECT): (M)(NREC) \]

\[ I = I + 1 \]

Return
# VARIABLES

<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE*</th>
<th>No. of Words</th>
<th>INPUT/TEMP OUTPUT</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>I</td>
<td>T</td>
<td>NRPC</td>
<td>1</td>
<td></td>
<td>Used as index in Read/Write loop</td>
</tr>
<tr>
<td>J</td>
<td>I</td>
<td>T</td>
<td>2560</td>
<td>1</td>
<td></td>
<td>Used in indexed Read/Write</td>
</tr>
<tr>
<td>L</td>
<td>I</td>
<td>T</td>
<td>2560-NRL+1</td>
<td>1</td>
<td></td>
<td>Beginning of current record being read/written</td>
</tr>
<tr>
<td>M</td>
<td>I</td>
<td>I</td>
<td>+1</td>
<td>-1</td>
<td></td>
<td>M=-1 use top half of NARAY; M=+1 use bottom half</td>
</tr>
<tr>
<td>N</td>
<td>I</td>
<td>I</td>
<td>2560</td>
<td>NRL</td>
<td></td>
<td>End of current record being Read/Written</td>
</tr>
<tr>
<td>NARAY</td>
<td>I</td>
<td>I</td>
<td>520</td>
<td></td>
<td></td>
<td>Name of array for Record Storage (RSA)</td>
</tr>
<tr>
<td>NF</td>
<td>I</td>
<td>I</td>
<td>32767</td>
<td>1</td>
<td></td>
<td>File number to be sorted</td>
</tr>
<tr>
<td>NREC</td>
<td>I</td>
<td>I</td>
<td>32767</td>
<td>1</td>
<td></td>
<td>Current record number for I/O</td>
</tr>
<tr>
<td>NRECT</td>
<td>I</td>
<td>I</td>
<td>32767</td>
<td>1</td>
<td></td>
<td>Last record number in this pass</td>
</tr>
<tr>
<td>NRL</td>
<td>I</td>
<td>I</td>
<td>320</td>
<td></td>
<td></td>
<td>Record Length</td>
</tr>
<tr>
<td>NRPO</td>
<td>I</td>
<td>I</td>
<td>1280</td>
<td>B</td>
<td></td>
<td>Number of records per cylinder</td>
</tr>
<tr>
<td>NRW</td>
<td>I</td>
<td>I</td>
<td>2</td>
<td>I</td>
<td>2</td>
<td>Read/Write Switch. Read=1, Write=2</td>
</tr>
</tbody>
</table>

*Mode: I = integer, R = real, D = decimal, A = alphabetic.
NEXS = 0
J = LARRY - NRL

NEX = 0
K = 1

NCOMP
NARAY, K + KS - 1,
K + KE - 1, NARAY,
K + NRL + KS - 1

NCOMP > 0

MOVE
NARAY, K,
K + NRL - 1, ITEM 1

MOVE
NARAY, K + NRL,
K + NRL + NRL - 1,
NARAY, K

MOVE
ITEMP, 1, NRL,
NARAY, K + NRL

NEX = NEX + 1

K = K + NRL

J = J - NRL
NEXS = NEXS + NEX

RETURN
<table>
<thead>
<tr>
<th>NAME</th>
<th>MODE</th>
<th>No. of Words</th>
<th>Input</th>
<th>Output</th>
<th>MAX. VALUE</th>
<th>MIN. VALUE</th>
<th>FUNCTION OF VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>*ITEMP</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A temporary area for storing one record during an exchange</td>
</tr>
<tr>
<td>J</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>4800</td>
<td></td>
<td>The last record out of order</td>
</tr>
<tr>
<td>K</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>Used as index in loop</td>
</tr>
<tr>
<td>*KE</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>End of the control key, its ending position</td>
</tr>
<tr>
<td>*KS</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>Beginning of the control key, its starting position</td>
</tr>
<tr>
<td>LARRY</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>5120</td>
<td></td>
<td>Length of the Record Storage Area (RSA)</td>
</tr>
<tr>
<td>MOVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A subroutine to move a record</td>
</tr>
<tr>
<td>*NARAY</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Name of array for Record Storage (RSA)</td>
</tr>
<tr>
<td>NCOMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A subroutine to compare two control keys</td>
</tr>
<tr>
<td>NEX</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>A count of the number of exchanges during a single two-cylinder sort pass</td>
</tr>
<tr>
<td>*NEXS</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>A count of the total number of exchanges during</td>
</tr>
<tr>
<td>NRL</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>320</td>
<td>2</td>
<td>The record length</td>
</tr>
</tbody>
</table>

*Modal: I = integer, R = real, D = decimal, A = alphabetic

Calling Parameters
SUMMARY

Generally there are two approaches to sorting with your 1130:
1. Avoid sorting
2. Write a sort program
The ways in which you can avoid sorting are:
1. Maintain multiple copies of your files.
2. Maintain multiple copies of your index, if an index exists.
3. Sort offline.
   If you decide that sorting is necessary, many techniques are available. The methods available and a brief evaluation of each were given earlier.
Section 80: USE OF THE DISK FOR DATA STORAGE

CONTENTS

General ............................... 80.01.00
The Physical, or Hardware, Structure of the Disk ......................... 80.10.00
The Disk as Seen by the FORTRAN Programmer ................................. 80.20.00
The Interrelationship of the Physical and Logical Structures ............... 80.30.00
The DEFINE FILE Statement ........... 80.30.10
The *STOREDATA and *FILES Cards .......... 80.30.20

Record Lengths and Sector Utilization .. 80.40.00
A Trick to Get Long Records and/or Better Packing ......................... 80.40.10
Computing Record Length ................. 80.50.00
Shortening Record Length ................ 80.60.00
Some Examples of Disk File Setup ......... 80.70.00
Example 1 ............................. 80.70.10
Example 2 ............................. 80.70.20
Example 3 ............................. 80.70.30
GENERAL

To make effective use of your disk storage capability, you need to know the way the disk is organized and the way your data will be set up on it. This section deals exclusively with the use of the disk as a data storage device. Although it is desirable (and often necessary) to store programs and subprograms on the disk, these normally present little difficulty, since the 1130 Monitor system handles most of the details involved.

The way in which the disk is used can significantly affect:
1. The amount of data that can fit into the available disk space
2. The running speed of programs using disk data files
3. The basic practicality of many jobs. (An improperly organized disk file can make the space and time requirements of some jobs appear excessive, when in reality they need not be.)
THE PHYSICAL, OR HARDWARE, STRUCTURE OF THE DISK

Each IBM 2315 disk cartridge contains 512,000 words organized into 200 cylinders of eight sectors each: a sector, in turn, contains 320 words (see Figure 80.1). This is a very rigid organization dictated by the basic design of the 1130.

A word

A sector, 320 words + address

A cylinder, 2 tracks, 8 sectors

A cartridge, 200 cylinders 512,000 words 1600 sectors

An entire cylinder (eight sectors) is accessed by one setting of the disk read/write heads. If you wish to read or write from a cylinder other than the one at which the heads are now set, the disk arm must be moved to the new cylinder. The disk mechanism moves the arm directly from the old position to the new position in steps of one or two cylinders. (It does not return to a "home" position first, as some other disk units do.) Both single steps and double steps take the same length of time: 15 milliseconds (.015 seconds). To move nine cylinders, you need four 2-cylinder moves (4x15 or 60 milliseconds) plus one 1-cylinder move (15 milliseconds) -- a total of 75 milliseconds. A move of ten cylinders takes the same amount of time -- five 2-cylinder moves (5x15 or 75 milliseconds). Figure 80.2 shows some representative arm movement times.

<table>
<thead>
<tr>
<th>Move This Many Cylinders</th>
<th>Seek Time</th>
<th>Stabilization Time</th>
<th>Average Rotational Delay Time</th>
<th>Read or Write</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>1 or 2</td>
<td>15</td>
<td>25</td>
<td>20</td>
<td>10</td>
<td>70</td>
</tr>
<tr>
<td>3 or 4</td>
<td>30</td>
<td>25</td>
<td>20</td>
<td>10</td>
<td>85</td>
</tr>
<tr>
<td>5 or 6</td>
<td>45</td>
<td>25</td>
<td>20</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>199 or 200</td>
<td>1500</td>
<td>25</td>
<td>20</td>
<td>10</td>
<td>1555</td>
</tr>
</tbody>
</table>

Figure 80.1. Disk storage definitions

Figure 80.2.
THE DISK AS SEEN BY THE FORTRAN PROGRAMMER

When programming in 1130 FORTRAN, the disk appears to be an entirely different device than the one just described. It consists of a data area which can be subdivided into any number of files, whose physical size, symbolic names, and symbolic numbers have been determined by you.

You may further subdivide each file into some number of equal-size blocks known as records. You choose the size of the record, and each record has a symbolic record number, starting with 1.

Within the record you can place fields, which may be real, decimal, or integer numbers, or alphameric data.

This is an extremely flexible system, as opposed to the rigid subdivisions and addresses of the actual hardware. It is still one and the same disk, however, and you must have a good knowledge of both systems to use the disk effectively. This section presents the basic guidelines by which you can relate these seemingly diverse systems:

The physical, or hardware, system
The logical, symbolic, or software system
THE INTERRELATIONSHIP OF THE PHYSICAL AND LOGICAL STRUCTURES

The DEFINE FILE Statement

For every data file you wish to access on the disk, there must be a DEFINE FILE statement in your FORTRAN program specifying certain details. A typical DEFINE FILE statement is

```
DEFINE FILE 47(400,85,U,NEXT)
```

which indicates a file numbered 47, having 400 records of 85 words each. The U is always required and specifies an unformatted record. NEXT is the name of an integer variable that will always be set to the record number of the next record in the file, a number between 1 and 400. For example, if you have just given the command

```
READ (47'K) A, B, I, J
```

where K was 96, NEXT will equal 97, the record number of the next record. The incrementing of NEXT occurs automatically and you may choose to ignore it completely. In this case, you are addressing your file by the symbol K, doing your own manipulation of K, and not using NEXT at all. If you wish to read the next record, you can say either

```
READ (47'NEXT) A, B, I, J
```

or

```
K = K + 1
READ (47'K) A, B, I, J
```

An 85-word disk record allows three records per sector (Figure 80.2), so that your file of 400 records will require 134 sectors (the exact answer, 133 1/3, must be adjusted upward to the next higher whole number).

(If your record length could somehow be shortened to 80 words, you could place four records per sector, reducing the sector requirement from 134 to 100, a substantial savings -- see 80.40.00.)

If you do not want to save this data file for use by a subsequent program, the DEFINE FILE statement is the only place you need reference it.

The DEFINE FILE statement specifies a mixture of physical (actual) and logical (symbolic) subdivisions:

- File number (symbolic)
- Number of records (symbolic)
- Number of words per record (actual)

Cylinders, sectors, and fields are nowhere mentioned.

The READ (or WRITE) statements specify only symbolic designations:

- File number (symbolic)
- Record number (symbolic)
- Field names (symbolic)
The *STOREDATA and *FILES Cards

In some programs, the DEFINE FILE statement is all that is required to specify the details of a data file. The file is placed in Working Storage (WS), and records may be read or written by that program and/or its subprograms.

If, on the other hand, you have a data file that is to be used by more than one program, you must do more than just specify it in a DEFINE FILE statement. You must get it out of Working Storage (WS) and into the User Area (UA) or Fixed Area (FX), where it will be protected from accidental destruction. Working Storage is true to its name: it is strictly a work area. Data placed in WS may still be there when it is needed; then again, it may not, since the IBM compilers, DUP, and other programs all use WS.

Working with data files in the User Area or Fixed Area requires the use of two additional cards: the *STOREDATA card and the *FILES card.

The *STOREDATA card is used first to create an entry in the LET or FLET, specifying the name, location, and size of your data file. In this case the *STOREDATA card, despite its name, does not really store your data; actually, your data has not even been written yet.

In the preceding example (file 47 -- 400 records of 85 words each, requiring 134 sectors) you must run a disk utility job

```
3. The *STOREDATA card contains mixed type information -- both actual (number of sectors) and symbolic (file name and cartridge identification number).

At this point you have run the *STOREDATA job, specifying certain data about the file named PAYRL, and compiled your FORTRAN program referencing the file numbered 47. How, when, and where can you tell the 1130 that these two files are one and the same?

How? With the *FILES card, which in this case would read

```

When and where? This depends on the format (DSF or DCI) of your program. If the program is in DSF format, you must place the *FILES card after the execute card every time you execute the program.

```

If the program is to be built into core image format (DCI), the *FILES card must be placed after the *STORECI card

```

and not after the // XEQ card.
As mentioned earlier, data may be placed in Working Storage (WS) if you do not intend to save it -- that is, if it is to be used for temporary storage within one JOB. In fact, data will be written in WS unless a *FILES card is used; likewise, any READ commands will assume that the data is in WS if no *FILES cards are present.

Using the *STOREDATA and *FILES combination, however, you have a choice as to where your data file will be placed -- either in the User Area (UA) or Fixed Area (FX). In most cases it does not matter which is chosen, since both areas are safe from accidental destruction. The main difference is that files in the Fixed Area are in a fixed position and will not be moved about as other files and programs are deleted.

This option is exercised by the characters punched in columns 17 and 18 of the *STOREDATA card -- UA indicating User Area, FX indicating Fixed Area. Columns 13 and 14 always contain WS, since the *STOREDATA always takes some number of sectors from WS and adds them to UA or FX.

Note that there is no Fixed Area on a disk cartridge unless you have defined one with a *DEFINE FIXED AREA card.
RECORD LENGTHS AND SECTOR UTILIZATION

Remember, the disk is physically composed of sectors, each containing 320 words. A symbolic record may not cross the boundary between two physical sectors; in other words, a record must lie entirely within one sector of 320 words. This means that a record cannot exceed 320 words in length. (Actually, it is possible to have records longer than 320 words, using a trick covered in a later subsection.) It does not mean that only one record may occupy a sector; it is possible that many records will be placed on one sector. For example, if your record size is twelve words, you may place 26 records onto each sector (26x12 = 312 words), with eight words (320-312) words remaining. These eight words will not be used for two-thirds of the 27th record, since that would violate the rule spelled out above. The remaining eight words will not be used, and are inaccessible to the FORTRAN programmer.

It goes without saying that you will gain the most efficient use of your disk if you utilize all 320 words of every sector. As the previous example shows, however, this may not always occur. Figure 80.3 shows the relationship between record size and sector utilization.

Clearly, certain record lengths result in very poor disk utilization. Take a 65-word record, for example. It will allow four records per sector, using 4x65 or 260 words, but leaving the remaining 60 words (about 20% of the sector) unused. On the other hand, if you could reduce the length of that record by one word, to 64, you could fit five records in a sector, using 5x64 or 320 words, and wasting none.

Inefficient use of the disk can have two major effects on your overall system:

1. A given number of records may require more space than is available on the disk. If you have 800 employee records at two per sector, you need 400 sectors or 50 cylinders, fully 25% of the disk. If you could fit three records per sector, your total sector requirements would drop to 274, or 34 cylinders. It is entirely possible that there are 30 cylinders available on a particular disk, but not 50.

In this case you either have to abandon the job, delete something else from the disk, or shorten the record size.

2. Even if 50 cylinders are available, you cannot escape the fact that you are using them inefficiently. If your 800 employee records are spread out over 50 cylinders, rather than 30, you will spend proportionately more time in disk arm movement. Your records will be 6% further apart, and your disk arm seek time will be about the same percentage greater.

Thus you have two incentives to make your disk records as short as possible. Several techniques for doing so are given in the subsection 80.60.00.

For long records (46 words or more), you should inspect your record size to determine whether it is at or slightly above a boundary, or break point -- 46, 54, 65, 81, 107, or 161 words. (See Figure 80.3.) If this is the case, it is worth considerable effort to shorten this record enough to increase the "packing factor" by one.

For medium records (19 to 45 words in length) the record size is always near a boundary or break point, so the packing factor can be increased by one or two with a small reduction in record length. With records of this length, however, it becomes more difficult to find ways to shorten the records.

For short records (9 to 18 words in length), even greater improvements are theoretically possible, but are proportionately more difficult to obtain.

NOTE: When shortening disk record lengths, always keep future needs in mind.

<table>
<thead>
<tr>
<th>Record Length</th>
<th>Records per Sector</th>
<th>Record Length</th>
<th>Records per Sector</th>
<th>Record Length</th>
<th>Records per Sector</th>
<th>Record Length</th>
<th>Records per Sector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>320</td>
<td>2</td>
<td>50</td>
<td>4</td>
<td>80</td>
<td>7</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>160</td>
<td>11</td>
<td>22</td>
<td>4</td>
<td>80</td>
<td>6</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>106</td>
<td>12</td>
<td>20</td>
<td>4</td>
<td>80</td>
<td>5</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td>13</td>
<td>24</td>
<td>3</td>
<td>65</td>
<td>4</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>14</td>
<td>22</td>
<td>3</td>
<td>65</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>53</td>
<td>15</td>
<td>21</td>
<td>3</td>
<td>65</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>46</td>
<td>16</td>
<td>20</td>
<td>3</td>
<td>65</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>17</td>
<td>18</td>
<td>3</td>
<td>65</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>35</td>
<td>18</td>
<td>17</td>
<td>3</td>
<td>65</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>

Figure 80.3.
A Trick to Get Long Records and/or Better Packing

If you have records exceeding 320 words in length, or records of a length that yields very poor packing, you may wish to employ a trick, or, more properly, an unorthodox usage of FORTRAN. This usage takes advantage of the fact that an 1130 FORTRAN READ/ WRITE I/O list will be satisfied, regardless of the FORMAT or DEFINE FILE statement.

For example, if we say

```
DIMENSION ITEMS(500)
... READ (6'N) ITEMS
```

500 ITEMS will be read from the disk, starting at record number N. It would not matter if the statement:

```
DEFINE FILE 6(100, 100, U, NEXT)
```

were used, indicating 100-word records. In fact, no matter what the DEFINE FILE statement contains, the entire ITEM array will be read, whether it exceeds 320 or not.

The DEFINE FILE statement has one effect, however, in that it still defines the length of the "defined" record at 100 words. Reading the 500-word array merely means that we have read five "defined" records. If N were 100, you would have to increment it by five to read the next 500 words or block of five 100-word records.

The DEFINE FILE statement, then, must define:

1. A file that contains enough space to hold all the data to be placed in it
2. A record length less than 320
3. Preferably, a record length evenly divisible into 320 -- that is, 320, 160, 80, 64, 40, 32, 20, 16, 10, 8, 5, 4, 2, 1.

To illustrate, suppose you have a file containing 100 records, each with 400 words. Since no matter what the DEFINE FILE statement contains, the entire ITEM array will be read, whether it exceeds 320 or not.

The reason for the third rule should be self-evident in light of the previous material in this section:

- The FILE 2 combination (200, 200) results in only one record per sector, with 320-200 or 120 words on each wasted. Total file size would be 200 sectors.
- FILE 3 (400, 100) gives three records per sector, with 320 -3 x 100 or 20 words wasted. Total file size is 400/3 or 134 sectors.
- FILE 4 (800, 50) yields six records per sector, with 320 -6 x 50 or 20 words wasted. Total file size is also 134 sectors.
- FILE 5 (500, 80) results in four records per sector, with 320 -4 x 80 or no words wasted. Total file size is 500/4 or 125 sectors.

To implement this trick, you need change only the DEFINE FILE statement and the incrementing/decrementing logic in existing programs. For example, if you have a file that formerly contained 400 records of 196 words each:

```
DEFINE FILE 6 (400, 196, U, NEXT)
```

you now realize that it will use each sector quite inefficiently. Therefore, you choose instead to use

```
DEFINE FILE 6 (1600, 50, U, NEXT)
```

which replaces the old 196-word record with four 50-word records. (In addition to better packing, you gain four words in each record.) In the body of your program, where you coded

```
N = N + 1
READ (6'N) data
```

you use instead

```
N = N + 4
```

and so on throughout the program.

Where you use the automatically incremented parameter NEXT:

```
READ (6'NEXT) data
```

you need do nothing; NEXT will automatically reflect the number of defined records that have been processed, and will be incremented by 4 rather than 1.
COMPUTING RECORD LENGTH

Once you have decided what data will be included in your disk record, you may easily calculate the length of the record by listing the fields in the record, totalling the number of real fields (called R), the number of integer fields (called I), and using the table below to determine the number of words:

<table>
<thead>
<tr>
<th>*EXTENDED PRECISION Card Used</th>
<th>No Precision Card Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>*ONE WORD INTEGERS Card Used</td>
<td>WORDS = (3xR)+I</td>
</tr>
<tr>
<td></td>
<td>WORDS = (2xR)+I</td>
</tr>
<tr>
<td>No Integers Card Used</td>
<td>WORDS = (3xR)+(3xI)</td>
</tr>
<tr>
<td></td>
<td>WORDS = (2xR)+(2xI)</td>
</tr>
</tbody>
</table>

In the case of fields comprising the items of an array (often alphanumerical data), the number of items is the size of the array. For example, if you have a field consisting of a 16-character name, placed two characters per word (A2 format) in the array NAME, this will count as eight items rather than one, when you are calculating I.

The task is simplified by the fact that *ONE WORD INTEGERS should always be used, reducing all integers to one word per field. Except in very unusual cases, you should compile all of your programs using the *ONE WORD INTEGERS control card.
SHORTENING RECORD LENGTH

The following suggestions will help you shorten the length of disk records. The first three should be taken regardless of record length, since they represent good programming practice and involve little or no effort. Suggestions 4-7 involve more programming effort and core storage. You must determine how much effort it is worth to gain more space on the disk.

1. First and foremost, each item in the disk record should be inspected to determine whether it really must be in this record. Can it be eliminated entirely, or placed in a separate file?

2. You should decide whether standard or extended precision should be used. The decision is usually based on other considerations; extended precision is normally used, but it does no harm to re-ask this question.

3. You should make certain that the *ONE WORD INTEGERS control record is included when compiling all programs. If not, each integer will occupy two or three words, depending on the use of standard or extended precision.

4. Each real (floating point) field should be studied with an eye toward converting it to integer mode. Remember, in most cases integers require only one word; real fields, three words. Integers are limited to a magnitude of 32767, but many items in your application may never exceed this limit, which may be thought of as

- 32767 units, pieces
- 327.67 dollars, hours, percent
- 3.2767 pay rate, etc

where the decimal points are implied and handled by you. Some typical items that lend themselves to such treatment are:

- Discount and interest rates
- Prices or price differentials
- Inventory -- quantity on hand, quantity on order, etc.
- Payroll -- savings bond deduction, city and state taxes, miscellaneous deductions

5. Alphabetic data should be placed on the disk with two characters per word (A2) rather than one (A1). In many cases, data (numeric and alphabetic) will be read from a card using A1 format (one character per word) for later processing with the Commercial Subroutine Package. Before this data is written on the disk, it may be compressed into two-characters-per-word format (A2), using the PACK subroutine supplied with CSP. Typical items in this category include names and addresses, descriptions, Social Security numbers, etc.

6. If necessary, three alphabetic characters can be placed in one word for disk storage. This can be done by subroutine that involves a table of 40 EBCDIC codes and a packing/unpacking formula. Only 40 characters are allowed -- for example:

0123456789ABC ....................XYZb--,

where b signifies a blank.

If you have just read the three characters B2- (called LTR1, LTR2, and LTR3 respectively) from a card with A1 format, the subroutine can look up their positions in the table and find that:

LTR1, a B, is 12th
LTR2, a 2, is 3rd
LTR3, a -, is 38th

Then, using the formula

\[
INA3 = LTR3 + 40\times LTR2 + (LTR1-20)\times 1600
\]

or \(INA3 = 38 + 40\times 3 + (12-20)\times 1600\)

the subroutine obtains -12642.

This is a unique representation of the three characters B2-, and can be placed on the disk as one word.

To decode after reading from the disk, the subroutine manipulates INA3 to obtain LTR1, LTR2, and LTR3:

\[
LTR1 = \begin{cases} 
(32,000 + INA3)/1600 & \text{if negative} \\
INA3/1600 + 20 & \text{if positive} 
\end{cases} = 12
\]

\[
LTR2 = (INA3 - 1600) \times (LTR1-20)/40 = 3
\]

\[
LTR3 = (INA3 - 1600) \times (LTR1-20) - 40\times LTR2 = 38
\]

Looking up these three codes from the same table, you may return to A1 format.
7. In many cases, several values may be combined into one word. For example, in a payroll file, you might have four different variables:

- **IE**: (Exempt or nonexempt)
  - 1 = EXEMPT
  - 2 = NONEXEMPT
- **MSC**: (Marital Status Code)
  - 1 = SINGLE
  - 2 = MARRIED
- **MORF**: (Male OR Female)
  - 1 = MALE
  - 2 = FEMALE
- **NDEP**: (Number of DEPendents)
  - 0 through 99

One way to compress these four items into a five-digit word (called KODE) is:

<table>
<thead>
<tr>
<th>Digit</th>
<th>Description</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Exempt or nonexempt</td>
<td>IE</td>
</tr>
<tr>
<td>2</td>
<td>Marital status code</td>
<td>MSC</td>
</tr>
<tr>
<td>3</td>
<td>Male or female</td>
<td>MORF</td>
</tr>
<tr>
<td>4, 5</td>
<td>Number of dependents</td>
<td>NDEP</td>
</tr>
</tbody>
</table>

For example, if KODE = 22103, this employee is:

- Nonexempt (digit 1 is 2)
- Married (digit 2 is 2)
- Male (digit 3 is 1)
- With three dependents (digits 4 and 5 are 03)

To **compress** these values before writing on the disk, all you need do is:

\[
KODE = (IE \times 10000) + (MSC \times 1000) + MORF \times 100 + NDEP
\]

To decompress the word KODE after reading it from the disk, you could use a function similar to the one below, called `NDIG`:

```fortran
FUNCTION NDIG (N, IT)
DIMENSION IZ(6)
DATA IZ/32767, 10000, 1000, 100, 10, 1/
NDIG = IT/IZ(N+1) - IT/IZ(N)*10
RETURN
END
```

Using this function:

- IE = NDIG (1, KODE)
- MSC = NDIG (2, KODE)
- MORF = NDIG (3, KODE)
- NDEP = NDIG (4, KODE) \times 10 + NDIG (5, KODE) etc.

In this case, such a packing technique will save three words on each disk record (by using one word rather than four). This may or may not be worth the added programming involved, the additional core storage required for the function, and the packing/unpacking coding.

Don't forget that KODE is an integer, and its magnitude is limited to 32767. To be safe, you should plan for a limit of 29999 for such compressed words.
SOME EXAMPLES OF DISK FILE SETUP

Example 1.
A program reads a deck of cards and builds two large tables of data. The individual data items are of no particular interest; however, after the last input card has been processed, you want to summarize the data tables and print a summary report. The data tables required are so large that they cannot fit in core storage; therefore, you decide to use the disk as an extension of core storage to accumulate the two tables.

After this job has been run, you have no need for the data, so you decide to keep it in Working Storage (WS).

Two files are required, numbered 1 and 2, and any disk cartridge may be used. Each of the two files contains 100 records of 150 words each.

Since the files are of a temporary nature and will remain in Working Storage, neither a *STORE-DATA card nor a *FILES card is required; consequently the files have no names, only numbers. The next two exhibits show how the Disk File Layout Worksheet would be filled in for these two files.
### DISK FILE LAYOUT WORKSHEET

#### TABLE OF SALES BY ITEM, CLASS

<table>
<thead>
<tr>
<th>File Number</th>
<th>File Name</th>
<th>Cartridge ID Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description of File**

*STORE DATA card

<table>
<thead>
<tr>
<th>File Name</th>
<th>Cartridge ID Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Next Indicator**

*FILES (11 OR *FILES *)

OR *FILES If ID number is not used

**Number of Words per Record**

150

**Number of Records**

100

**Define File**

2 (100, 150, N)

---

<table>
<thead>
<tr>
<th>Record Length</th>
<th>Records per Sector</th>
<th>Records per Sector</th>
<th>Records per Sector</th>
<th>Records per Sector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>160</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>196</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>53</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>45</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>38</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>

**Number of Records**

- File to be placed in:
  - WS
  - UA
  - FX

**Number of Records per Sector**

- Rounded Upward

---

*STORE DATA W S

<table>
<thead>
<tr>
<th>File Name</th>
<th>Number of Sectors</th>
<th>Cartridge ID Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 2.

A payroll and project cost accounting system involves four disk data files:

- **EMPS** - 300 employee records with 60 words per record.
- **PROJ** - 100 project records with 20 words per record.
- **DDESC** - 20 records of 18 words each, containing some alphabetic information for each of 20 departments.
- **SUBT** - 20 records of 60 words each, containing an array of subtotals of department worked for vs department charged to. This is a temporary file used only as an extension of core storage, not saved from job to job.

Assume you have only one disk drive and don't care which disk cartridge is mounted. (You really do, but you, rather than the Monitor, will make sure the correct disk is being used.)

With this basic data, you can fill in the Disk File Layout Worksheets and punch the necessary cards. Note that file 9 (SUBT) does not need a FILES OR STOREDATA card, since it is not to be saved from one job to another. It does require a DEFINE FILE card.
**DISK FILE LAYOUT WORKSHEET**

**MA S T E R P R O J E C T C O S T S**

<table>
<thead>
<tr>
<th>Description of File</th>
<th>File Number</th>
<th>File Name</th>
<th>Cartridge ID Number</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STORE DATA</strong></td>
<td>74</td>
<td>PROJ</td>
<td></td>
</tr>
</tbody>
</table>

**Next Indicator**

**Number of Words per Record**

**Number of Records**

1000

**DEFINE FILE**

1000 22 U N

**Number of Records**

1000

**Number of Records per Sector**

14

\[ 1000 \div 14 = 71.5 \]

**Rounded Upward**

**WS**

**UA**

**FX**

**Don't need *STORE DATA card**

**STORE DATA**

<table>
<thead>
<tr>
<th>13 14 17 18 21 22 23 24 25 27 28 29 30</th>
<th>37 38 39 40</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>Number of Sectors</td>
</tr>
</tbody>
</table>

**File to be placed in:**

WS UA FX
**DISK FILE LAYOUT WORKSHEET**

### File Number
- **9**

### File Name
- **SUBT**

### Cartridge ID Number
- Blank

**Next Indicator**
- **W 9**

**Number of Words per Record**
- **60**

**Number of Records**
- **20**

**File to be placed in:**
- **WS UA FX**

### Record Lengths

<table>
<thead>
<tr>
<th>Record Length</th>
<th>Records per Sector</th>
<th>Records per Cartridge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>250</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>160</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>90</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

**Don't need *STORE DATA card**

**Rounded Upward**

**Number of Records**

\[ 20 \div 5 = 4 \]

**Number of Records per Sector**

**Record Length**

<table>
<thead>
<tr>
<th>Records per Sector</th>
<th>Record Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

**File Name**

**Number of Sectors**

**Cartridge ID Number**
Example 3.

This is the same job as in example 2, except that two disk drives are now available and you are going to be more careful about which disk cartridge is mounted on which drive unit:

<table>
<thead>
<tr>
<th>DRIVE 0</th>
<th>DRIVE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARTRIDGE ID 0012</td>
<td>CARTRIDGE ID 0019</td>
</tr>
<tr>
<td>WS</td>
<td>UA</td>
</tr>
<tr>
<td>EMPS</td>
<td>SUBT</td>
</tr>
</tbody>
</table>

First, you have decided that cartridge 0012 will be on drive 0 and cartridge 0019 on drive 1. How is this communicated to the Monitor? It is done with the // JOB card, which must be punched:

```
// JOB 00/2 00/9
```

File SUBT, since it is in WS, really does not need a name (or a *FILES or *STOREDATA card). How, then, can you tell the 1130 Monitor on which disk drive it should be placed? Again, with the // JOB card. Columns 41-44 should contain the cartridge ID number of the disk to be used for Working Storage, in this case 0019.

The special // JOB card

```
// 108 00/2 00/9 GG/9
```

must be used when running the DUP *STOREDATA jobs and when executing the programs that use these files. Needless to say, the disk cartridges should be placed in the proper disk drive units.

The remaining three files (EMPS, PROJ, AND DDESC) are handled in a different manner. Since they are to be in the UA, they do require *FILES and *STOREDATA cards, which contain a field for placing the cartridge ID number. These are shown on the Disk File Layout Worksheets following.
DISK FILE LAYOUT WORKSHEET

**EMPLOYEE RECORDS**

<table>
<thead>
<tr>
<th>Description of File</th>
<th>File Name</th>
<th>Cartridge ID Number</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EMPLOYEE RECORDS</strong></td>
<td><strong>EMPS</strong></td>
<td><strong>0012</strong></td>
</tr>
</tbody>
</table>

**File Number**: 6

**Number of Records**: 300

**Next Indicator**

**Number of Words per Record**: 60

**Define File**: Segment: 6, Subsections: 30, Page: 02

**Record Length**

- 1: 320
- 2: 160
- 3: 106
- 4: 80
- 5: 64
- 6: 53
- 7: 45
- 8: 40
- 9: 34
- 10: 28
- 11: 24
- 12: 20
- 13: 17
- 14: 16
- 15: 15
- 16: 14
- 17: 13
- 18: 12
- 19: 11
- 20: 10
- 21: 9
- 22: 8
- 23: 7
- 24: 6
- 25: 5
- 26: 4
- 27: 3
- 28: 2
- 29: 1
- 30: 0

**Records per Sector**

- 1: 32
- 2: 16
- 3: 10
- 4: 8
- 5: 6
- 6: 5
- 7: 4
- 8: 3
- 9: 2
- 10: 1
- 11: 0

**Number of Records in a File**

- **300**

**Number of Records per Sector**

- **60**

**File to be placed in**

- **WS 0A FX**

**Don't need *STORE DATA card**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Number of Sectors</th>
<th>Cartridge ID Number</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EMPS</strong></td>
<td><strong>60</strong></td>
<td><strong>0012</strong></td>
</tr>
</tbody>
</table>

**Rounded Upward**: 60

**Number of Records**

- 300

- **5**

- **60**

**File to be placed in**: Segment: 6, Subsections: 30, Page: 02
**DISK FILE LAYOUT WORKSHEET**

**MASTER PROJECT COSTS**

<table>
<thead>
<tr>
<th>Description of File</th>
<th>File Number</th>
<th>File Name</th>
<th>Cartridge ID Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>74</td>
<td>PROJ</td>
<td>0019</td>
</tr>
</tbody>
</table>

*FILES* (projection, 0019) or *FILES* (projection, 0019) if ID number is not used.

**Next Indicator**

**Number of Words per Record**

**Number of Records**

1000

**DEFINE FILE**

1000, 22, U, N

**File to be placed in:**

WS, UA, FX

Don't need *STORE DATA card.

**File Name**

WS, UA, PROJ, 72

Cartridge ID Number

0019

**Number of Records**

1000

**Number of Records per Sector**

14

**Rounded Upward**

71.5

**Record Length**

Records per Sector

---

<table>
<thead>
<tr>
<th>Record Length</th>
<th>Records per Sector</th>
<th>Records per Sector</th>
<th>Records per Sector</th>
<th>Records per Sector</th>
<th>Records per Sector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>300</td>
<td>10</td>
<td>32</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>160</td>
<td>11</td>
<td>29</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>13</td>
<td>12</td>
<td>24</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td>13</td>
<td>13</td>
<td>24</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>14</td>
<td>22</td>
<td>22</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>53</td>
<td>15</td>
<td>21</td>
<td>27</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>46</td>
<td>16</td>
<td>20</td>
<td>30 - 32</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>17</td>
<td>16</td>
<td>33 - 36</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>36</td>
<td>18</td>
<td>17</td>
<td>36 - 40</td>
<td>9</td>
</tr>
</tbody>
</table>

**File Name**

WS, UA, PROJ, 72

Number of Sectors

27 28 29 30

Cartridge ID Number

0019
DISK FILE LAYOUT WORKSHEET

FILE DESCRIPTIONS OF DEPARTMENTS

File Number: 33
File Name: DES
Cartridge ID Number: 0019

*FILES

OR *FILES

If ID number is not used

Next Indicator: 113
Number of Words per Record: 18
Number of Records: 20

DEFINE FILE

File to be placed in:
WS UA FX

Number of Records: 20
Number of Records per Sector: 17

File Name: DES
Number of Sectors: 2
Cartridge ID Number: 0019

Don't need *STORE DATA card
### DISK FILE LAYOUT WORKSHEET

**Description of File**

**SUBTOTALS**

<table>
<thead>
<tr>
<th>File Number</th>
<th>File Name</th>
<th>Cartridge ID Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>SUBT</td>
<td>0015</td>
</tr>
</tbody>
</table>

- **Next Indicator**: 9
- **Number of Words per Record**: 60
- **Number of Records**: 20

#### DEFINE FILE

- 9 (20, 60, 9)

#### Record Lengths and Records per Sector

<table>
<thead>
<tr>
<th>Record Length</th>
<th>Records per Sector</th>
<th>Record Length</th>
<th>Records per Sector</th>
<th>Record Length</th>
<th>Records per Sector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>320</td>
<td>10</td>
<td>22</td>
<td>10 - 30</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>160</td>
<td>11</td>
<td>29</td>
<td>21 - 30</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>12</td>
<td>26</td>
<td>22 - 30</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td>13</td>
<td>24</td>
<td>23 - 30</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>14</td>
<td>22</td>
<td>25 - 30</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>53</td>
<td>15</td>
<td>21</td>
<td>27 - 30</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>45</td>
<td>16</td>
<td>20</td>
<td>28 - 32</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>17</td>
<td>18</td>
<td>30 - 32</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>36</td>
<td>18</td>
<td>17</td>
<td>33 - 36</td>
<td>9</td>
</tr>
</tbody>
</table>

- **Number of Records**: 20
- **Number of Records per Sector**: 5
- **Rounded Upward**: 4

#### File to be placed in:

- WS
- UA
- FX

Don't need *STORE DATA card

<table>
<thead>
<tr>
<th>*STORE DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 14 17 18</td>
</tr>
</tbody>
</table>

- **File Name**
- **Number of Sectors**
- **Cartridge ID Number**
SECTION 85: DISK DATA FILES -- ORGANIZATION AND PROCESSING

CONTENTS

General ........................................ 85.01.00
Organization ................................. 85.10.00
  General ..................................... 85.10.01
  Pure Sequential ........................... 85.10.10
    Searching a Pure Sequential File
    Adding Items to the File
Indexed Sequential ......................... 85.10.20
    Choosing an Index Step Size
    Building the Index
    Searching the Index
    Maintaining the Index
    Adding Items to the File

Direct, or Random Organizations .......... 85.10.30
  Direct
  Computed Direct
  Partitioned Direct
Summary

Processing ................................... 85.20.00
  The Interaction of Organization and Processing .................. 85.30.00
    Introduction ................................ 85.30.01
    Choosing the Organization .............. 85.30.10
GENERAL

Data records should be filed according to a plan. The relationships between file organization and data processing should be carefully considered before this plan is chosen. With a disk, both storage and processing can be accomplished by either of two basic methods - sequential or direct (or random). Thus the following four storage-processing approaches are available:

- Sequential processing of sequentially organized data
- Random processing of sequentially organized data
- Sequential processing of randomly organized data
- Random processing of randomly organized data

The first two are the most commonly used approaches. The third and fourth are of limited use in most applications. However, the fourth offers some benefits (in selected applications), particularly when the data files undergo frequent additions and deletions, or when most of the transactions must be processed randomly.
ORGANIZATION

General

In a sequentially organized file, records are stored on the disk in control key sequence, so that records with successively higher control keys have successively higher record numbers. It is not necessary (or customary) for the control key to be the same number as the record number. The only requirement is that the control keys be in sequence, and in sequential (not necessarily consecutive) locations. Often, to narrow the search for a record in a sequential file, an index is consulted for the record number. This index is a sequential list of the keys of selected data file records with their corresponding record numbers. An example of a sequentially organized data file is a telephone directory, in which people are listed one after the other, in alphabetic order, the control key being the last name/first name combination, and the data being the telephone number.

In a randomly organized file, the records are generally stored in the sequence of their control keys. However, a mathematical transformation of the control key yields the record number. To find a record in such a file, the record number is computed from the control key by using the same transformation formula. In the random approach no index tables are required.
Pure Sequential

In a purely sequential disk data file, your records are placed on the disk in some logical order, with no attempt to organize them or to keep track of where they are placed. If a certain record is desired, the disk is searched sequentially until that record is found.

Searching a Pure Sequential File

Searching a pure sequential file is simple, but finding any particular record may be time-consuming in the case of large files. (If you are processing only a small number of records, however, the effect on the overall running time may be slight.) If you have a file of 1000 records, you can search for the item with key KEYXX, using the following FORTRAN statements:

```
DO 14 NREC=1,1000
   READ (NFILE'NREC) KEY
   IF (KEY-KEYXX) 14, 77, 14
14 CONTINUE
   ...
77 KEYXX has been found at record NREC.
```

KEY is the control key on the disk record, and KEYXX is the key you are searching for. If KEYXX is the 608th item, you will read, check, and get "no hit" on 607 items before reaching the 608th. A better way to search such a file is obvious: read every nth record until you pass the key being sought, then back up one record at a time until you find KEYXX.

```
DO 14 NREC=1,1000, NTH
   IREC = NREC
8  READ (NFILE'IREC) KEY
   IF (KEY-KEYXX) 14, 77, 66
66 IREC = IREC-1
   GO TO 8
14 CONTINUE
   ...
77 KEYXX has been found at record IREC.
```

If n is 20, you will read and check 32 records (1, 21, 41, 61, .... 601, 621) until you have passed the desired item (KEYXX, the 608th). Then 13 more records in the backing-up portion of the search (620, 619, 618, .... 609, 608) must be read. Here, the "skip" search has reduced your disk reads from 608 to 45, with a concurrent drop in processing time.

A further improvement can be made if you search first in large increments (say 100), then, when you pass the desired item, back up with a smaller increment (say 20) and, after passing the desired item the second time, switch to an increment of 1. Again, looking for the 608th item, the search will be - 1, 101, 201, 301, 401, 501, 601, 701, 681, 661, 641, 621, 601, 602, 603, 604, 605, 606, 607, 608, which involves 20 disk reads.

All the methods shown above, however, have one disadvantage: because they start at record number 1, the disk arm must move back to that record each time. A more elegant search technique would involve starting from wherever you found the last record, rather than from the beginning of the file. This assumes that the disk arm is still positioned over the last record, but it will not be so positioned if you have meanwhile used LOCAL or SOCIAL subroutines or accessed a record in another file on the same disk. This technique, therefore, is often impractical.

Another technique involves the method of halving, sometimes called a binary search. Suppose you have a file of 1000 records and you want to find the record whose key is KEYXX. First, halve the file size to obtain 500, and check the 500th record. If you do not find KEYXX there, halve the 500 to obtain 250 and, if the 500th record KEY was higher than KEYXX, check 500-250 or 250 next; if it was lower, check 500+250 or 750 next. The increment next becomes 125, then 63 (62.5 rounded upward), then 32 (31.5 rounded upward), etc. Using the previous example (KEYXX is the 608th item), your search pattern would have been:

```
500 first try
low, so +250
750 second try
high, so -125
625 third try
high, so - 63
562 fourth try
low, so + 32
594 fifth try
low, so + 16
610 sixth try
high, so - 8
602 seventh try
low, so + 4
606 eighth try
low, so + 2
hit 608 ninth try
```

A sequence of only nine disk reads.
Adding Items to the File

Adding new records to a sequential file involves some advance planning. If your employee file now consists of 188 employee records in man number sequence 018, 023, 067, 107, 109, 667, 691, 806, 902 where should you put the newest employee, who has just been assigned man number 098? You could rebuild the entire file, but that might prove time-consuming in the case of large files.

One way to handle file additions is to set up a separate "addition area" on the disk, either as a separate file or as a special area in the main file. With the latter option, new employees would be placed at the end of the file, starting with the last record and working backward.

For example, suppose the 188 employee records have been placed in a 200-record file. When man number 098 is added, it is placed in record number 200; the next new man number goes in 199; and so on.

The search programming becomes somewhat more involved: if a man number is not found in the main (sequential) portion of the file, the "addition area" is searched. If it is not found in either place, an error message is printed.

Since this added work will slow the running of the program, the file should be reorganized periodically, and new man numbers put in their proper places in the sequential file.

Programming such a search is easy:

```
NREC = 0
INC = 500
3  INC = (INC+1)/2
   READ (NFILE,NREC) KEY
   IF (KEY=KEYXX) 8,9,10
8  NREC = NREC + INC
   GO TO 3
10 NREC = NREC-INC
   GO TO 3
9  KEYXX has been found at record NREC.
```
Indexed Sequential

An indexed sequential file is essentially the same as a pure sequential file except that you maintain a table or index to the file, making it easier to find records. Suppose you have an inventory file containing 2500 items, with stock numbers ranging from 00001 to 28406. The stock number is kept as an integer, and the items have been placed on the disk in stock number sequence. In order to find an item on the disk, you will maintain an index consisting of the stock number of every 25th item. This will be a FORTRAN array in core storage. It will require 2500/25 or 100 entries in the index table:

INDEX (1) = 67, the stock number of the 25th item
INDEX (2) = 103, the stock number of the 50th item
INDEX (3) = 297, the stock number of the 75th item

... 

INDEX (99) = 28073, the stock number of the 2475th item
INDEX (100) = 28406, the stock number of the 2500th item

When it comes time to find an item on the disk, you first look for it in the core storage array INDEX. You probably will not find that particular item in the INDEX array, but you can get a good idea of its location. Suppose you have just read a card containing ITEM number 181. You look it up in the INDEX table as follows:

INDEX (1) = 67, which is lower than 181
INDEX (2) = 103, which is lower than 181
INDEX (3) = 297, which is higher than 181

The search stops here, since it is obvious that you have just passed item number 181 in the process of moving from INDEX (2) to INDEX (3). Since INDEX (2) is the 2x25 or 50th disk record and INDEX (3) is the 3x25 or 75th disk record, you know item 181 is between records 51 and 75.

Now resume your search for item 181, this time on the disk rather than in core. You may start at 51 and work your way up, or at 74 and back down. In the latter case, your program reads record 74, checks the stock number to see if it is 181, then reads record 73, 72, 71, 70, 69, 68, etc., down to record 51. If 181 is on the disk and in the right order, you will find it relatively quickly.

Choosing an Index Step Size

In the above example, 25 was arbitrarily chosen as the index step size; in other words, every 25th item in the file is recorded in the index table. What is the best index step size? First, for convenience, it should be an even divisor of the number of records in the file. If it is not, it complicates programming. Second, it should be about the same as or less than the number of records in a cylinder. For example, say your record size is 48 words. This allows six records per sector, and 8x6 or 48 records per cylinder. If you have 5000 records, you can choose 40 as your step size, making your INDEX array length 5000/40 or 125. The smaller the step size, the more likely you are to hit the right cylinder on the first disk arm movement. The probability that you will find the desired record on the first cylinder accessed is:

\[ 1 - \frac{(STEP \ SIZE - 1)}{(2 \times NO \ RECS \ PER \ CYL)} \]

or in this case:

\[ 1 - \frac{(40 - 1)}{(2 \times 48)} \]

or about 0.6.

In other words, with 48 records per cylinder, and an index of every 40th record, there are six chances in ten that the desired record can be found with one disk arm movement (seek), and four chances in ten that a second seek and read will be required. Such a second step will take about 65 milliseconds.

If you processed 225 inventory items, this second seek and read would add about one minute to the total running time of the job.

If you increase your step size to 50, the size of your index table in core drops from 125 to 100 items, but your probability of a second seek and read increases from .40 to .51.

On the other hand, if you decrease your step size to 25, your index table requires 200 entries, but your probability of a second seek drops to .25.
Building the index

Building your index of every 25th (or 90th, or whatever) item in your file presents no difficulty.

Option 1: Build the index at the same time that you load the data on the disk. All you need do is to keep a sequential number for each item (NO) and place its item number (or stock number, or employee number) in the INDEX array at position

\[ \text{NO} / (\text{ISS} + 1) + 1 \]

where ISS is the index step size. In FORTRAN, keeping an index of every four ITEMS (ISS=4) can be done like this:

```fortran
ISS = 4
NO = 1
55 READ (card) ITEM
   K = (NO-1) / ISS+1
   INDEX (K) = ITEM
   WRITE (file 'NO) ITEM, etc.
   NO = NO + 1
GO TO 55
```

Tracing through this coding, you will see that in addition to creating the data file on the disk:

<table>
<thead>
<tr>
<th>The ITEM number from this card</th>
<th>will be placed on this disk record</th>
<th>and at this position in the INDEX table</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>second</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>third</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>fourth</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>fifth</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>sixth</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>seventh</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>eighth</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>ninth</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When finished, the INDEX table will contain the ITEM numbers of the 4th, 8th, 12th, 16th, etc., records on the disk, just as desired. The INDEX can now be written on the disk as a separate file, for further use.

Option 2: Create an index file after the data records have been placed on the disk. This is even easier, since you need only read every 4th (or 20th, etc.) record from the disk and place its ITEM number in your INDEX table. Because this would be relatively slow, you would want to do it only once, with a separate program, storing the INDEX as a separate data file. Then, each program using the file could read it from the disk.
Searching the Index

Unlike pure sequential organization, which is searched on the disk, indexed sequential gives an index to search in core storage. The simplest approach is to search the table sequentially, one entry at a time, starting at the top. When you find an equal-or-less-than condition, you have found what you are looking for. The subroutine shown in

Figure 85.1 illustrates a typical method of searching an index.

You would CALL the subroutine FINDM with the known values of:

ITEM -- the item you are searching for
ITABL -- the name of the index table
LTABL -- the length of the index table
ISS -- the index table step size
NFILE -- the number of the file

```fortran
SUBROUTINE FINDM(NREC, ITEM, NFILE, ITABL, LTABL, ISS, IER)
DIMENSION ITABL(1)
IER = 1
DO 1 N = 1, LTABL
   IF ( ITEM - ITABL(N) ) 2, 9, 1
1 CONTINUE
   ITEM IS LARGER THAN THE LARGEST VALUE IN THE INDEX TABLE
   IER = 2
   RETURN
2 NREC = ISS * N
   DO 3 N = 1, ISS
   READ ( NFILE, NREC ) KEY
3 CONTINUE
   IF ( KEY - ITEM ) 4, 5, 6
4 NREC = NREC - 1
5 CONTINUE
   C ITEM IS NOT IN THE FILE IN THE AREA WHERE IT SHOULD BE
   IER = 3
   RETURN
5 CONTINUE
   C ITEM IS NOT IN THE FILE IN THE AREA WHERE IT SHOULD BE
   IER = 3
   RETURN
END
```

VARIABLE ALLOCATIONS

```
N(I) = 0000
KEY(I) = 0001
```

STATEMENT ALLOCATIONS

```
1 = 0033 2 = 0042 6 = 0058 3 = 0061 4 = 006A 5 = 006E
```

FEATURES SUPPORTED

```
TRANSFER TRACE
ARITHMETIC TRACE
ONE WORD INTEGERS
EXTENDED PRECISION
```

CALLED SUBPROGRAMS

```
SIAR SI IF SUBSC SUBIN SDRED SDI
```

INTEGER CONSTANTS

```
1 = 0004 2 = 0005 3 = 0006
```

CORE REQUIREMENTS FOR FINDM

```
COMMON 0 VARIABLES 4 PROGRAM 108
```

END OF COMPILATION
The subroutine returns:

NREC -- the record number where ITEM may be found
IER -- an error code:

1 -- ITEM has been found on the disk.
2 -- ITEM is larger than any entry in the index table.
3 -- ITEM is not on the disk where the index table indicates that it should be.

If IER is 2 or 3, the value of NREC returned is meaningless.

For example, suppose you have an index of 150 entries, called ITABL, representing every 60th item in an inventory file. After reading an inventory detail card containing a field called ITEM, you want to find the inventory record for that item. By using subroutine FINDM

CALL FINDM (NR, ITEM, NFILE, ITABL, 150, 60, IER)

you obtain, perhaps, an NR of 731 and an IER of 1, meaning that the desired ITEM has been found, at record 731. You can now read the inventory record for that item:

READ (NFILE'NR) data

Maintaining the Index

When using an indexed sequential disk data file, you must make sure that the index agrees completely with the file. If you rearrange records in the file without rebuilding the index, you may expect great difficulty in locating items in the file. Rebuilding the index is a rather simple matter, and two methods are given in a preceding section.

The file index is typically stored on the same disk as the file itself, and is read into core once, at the beginning of each program that uses the file it indexes.

Adding Items to the File

Adding items to an indexed sequential file can be handled in much the same manner as for pure sequential files. New records are placed in a separate file, or at the "high" end of the main file.

These new items will not be reflected in the index, but this does not matter too much. The index may be used to facilitate looking up records in the main portion of the file, and, if an item is not found there, it can be sought in the addition area.
Direct, or Random, Organizations

Direct

The simplest of all organizations exists when the record number is the same as the control key. For example, in a payroll application requiring one record per employee, the record number would be the same as the employee number. If you had a three-digit employee number, 001 to 999, you would set up a file of 999 records:

```
DEFINE FILE 1 (999, XXX, U, NEXT)
```

If you read an employee number from a card

```
77 FORMAT(I4)
READ (2,77) NEMP
```

you can immediately find that employee on the disk with

```
READ (1'NEMP) data
```

or

```
WRITE (1'NEMP) data
```

The advantages of this scheme are obvious, but the disadvantages may override them. In all probability, although there are 999 employee numbers, there are not really 999 employees, so there will be many "holes", or unused records, on the disk. Furthermore, 999 records, if they are large, may take up an inordinate amount of space on the disk. Even if they do fit on the disk, they will be spread out so far that programs using this file will run very slowly, because of the amount of "seeking", or disk arm movement, required.

One remedy would be to make the employee numbers more compact. If there are 300 employees, why not renumber them from 001 to 300? Or renumber your customers in a billing file? Or renumber your part numbers? Or job numbers?

Usually, this is more easily said than done, and you can expect difficulty in convincing management that they should change established systems just to make it easier for you or the computer.

Computed Direct

Sometimes it is possible to take an employee number (or part number, etc.) and modify it to make a usable record number. For example, if you have 300 employees with employee numbers between 3000 and 9000, you could take this number, NEMP, subtract 3000, divide by 20 (which is (9000-3000)/300), and add 1:

```
NREC = (NEMP-3000) / 20 + 1
```

This results in an NREC between 001 and 301. This is compact and wastes no space; however, two (or more) employee numbers may quite possibly result in the same record number. These are known as synonyms. There are many ways to handle this problem, but they require a certain added amount of programming and disk space.
Partitioned Direct

The disk addressing used by 1130 FORTRAN makes this method applicable in some cases. At one installation, there are about 150 employees, each with a four-digit employee number. The first two digits indicate the department number; the second two digits are sequential numbers. The distribution is as follows:

<table>
<thead>
<tr>
<th>Dept. No.</th>
<th>Maximum No. of Employees</th>
<th>Range of Employee Numbers</th>
<th>Number of Possible Employee Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>101 - 110</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>35</td>
<td>301 - 340</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>401 - 420</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>501 - 520</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>601 - 650</td>
<td>50</td>
</tr>
<tr>
<td>7</td>
<td>60</td>
<td>701 - 770</td>
<td>70</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1001 - 1020</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>20</td>
<td>1101 - 1130</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
<td>1201 - 1230</td>
<td>30</td>
</tr>
<tr>
<td>15</td>
<td>50</td>
<td>1501 - 1570</td>
<td>70</td>
</tr>
<tr>
<td>250</td>
<td></td>
<td></td>
<td>360</td>
</tr>
</tbody>
</table>

This user noticed that he could use this breakdown of employee numbers to advantage by setting up ten files:

```
DEFINE FILE 1 (10, X, U, N1)
DEFINE FILE 3 (40, X, U, N3)
DEFINE FILE 4 (20, X, U, N4)
DEFINE FILE 5 (20, X, U, N5)
DEFINE FILE 6 (50, X, U, N6)
DEFINE FILE 7 (70, X, U, N7)
DEFINE FILE 10 (20, X, U, N10)
DEFINE FILE 11 (30, X, U, N11)
DEFINE FILE 12 (30, X, U, N12)
DEFINE FILE 15 (70, X, U, N15)
```

requiring a total of 360 records to hold 250 employees. This wastes about one-third of the available records but results in much simplified programming, since the user can read the employee department and man number from a card:

```
77 FORMAT (I2, I2)
READ (2, 77) NDEP, NEM
```

and access that employee with a

```
READ (NDEP*NEM) data
```

statement.

Many numbering systems fit this general type and may lend themselves to this disk organization approach.

Summary

Each of the techniques described above has its advantages and disadvantages, as has been pointed out earlier. In general, indexed sequential files require more core and disk storage (because of the index) and tend to increase processing time because of the searching involved. Random (direct) organizations make for fast access, with little extra core or disk requirements, but are usually difficult to set up because of the synonym problem.
PROCESSING

Just as sequential and random are two basic ways to organize a file, they are also two ways to process or access a file.

If you process records in the same order as that in which they lie on the disk, you are processing sequentially; if you process in a different order, you are processing randomly. Thus the same two words (sequential and random) have substantially different meanings when used in the area of processing, since the definition of each depends on the organization of the file. This was not so when considering organization; a file was sequential or random depending on the order in which control keys were placed on the disk.

Consider the telephone directory -- a sequential file because the control keys (names) are in alphabetic order. If you scan through the directory, from front to back, looking for people who live on Main Street, or for men whose first name is John, you are processing sequentially, in the same order as the keys.

If you are looking for the telephone numbers of three friends -- J. DOE, P. ADAMS and L. SMITH -- and you look for them in that order (not alphabetic), you are processing randomly. On the other hand, if you sort them into alphabetic order -- ADAMS, DOE, and SMITH you are processing sequentially.
THE INTERACTION of ORGANIZATION and PROCESSING

Introduction

As you have seen, the two factors, organization and processing, are tied together quite intimately. Often, for this reason, it is not easy to make the basic decision as to which combination of techniques to use:

- Sequential organization, sequential processing
- Sequential organization, random processing
- Random organization, sequential processing
- Random organization, random processing

Actually, it is often impossible to use only one type. You can (and, perhaps, must) process in many sequences; but your file can have only one organization at any one time.
Choosing the Organization

Because of the interaction between processing and organization, there are few concrete guidelines for the user who must make this decision. However, the following outline will help lead the way toward one organization or the other. The payroll application is given as the example.

1. List the processing that must be done to this file and the required order of inputs and outputs (see Figure 85.2).

<table>
<thead>
<tr>
<th>Application</th>
<th>Required Order of:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INPUT</td>
</tr>
<tr>
<td></td>
<td>No order or doesn't matter</td>
</tr>
<tr>
<td>Edit input cards</td>
<td>Same as later process</td>
</tr>
<tr>
<td>Calculations</td>
<td>Employee number</td>
</tr>
<tr>
<td>Payroll register</td>
<td>Employee number</td>
</tr>
<tr>
<td>Payroll checks</td>
<td>Employee number</td>
</tr>
<tr>
<td>941 report</td>
<td>Employee number</td>
</tr>
<tr>
<td>Name and address stickers</td>
<td>✓</td>
</tr>
</tbody>
</table>

2. How many different sequences are there?
   a. None. No one really cares what the processing sequences are (order of card input, order of output on reports, etc.). Make sure this is so. If it is, go to step 3.
   b. One. There is only one basic processing sequence desired; go to step 4.
   c. More than one. This complicates the matter. Go to step 5. Processing sequences needed:
      1.
      2.
      3.
      4.
      5.

3. No one cares what the processing sequence is. This is unusual but does sometimes occur. If this is so, you can forget about processing, and choose an organization as an isolated problem, entirely separate from processing.

4. This file will never be processed in more than one sequence. Therefore, it would seem like a good idea to organize it either sequentially or randomly, in the same order as that required by processing.

5. This file must be processed in more than one order; however, it can be in only one order at any one time. Recheck step 1. Can any of the inputs be hand- or machine-sorted into the same order as another input? Can some of the output orders be relaxed? Can you somehow reduce the number of orders required? If you can reduce it to one, you can go to step 3 or 4. If not, you must sort your file from one order to the other, or otherwise work around this problem.
### Section 90: IMPROVING YOUR SYSTEM -- PERFORMANCE

**CONTENTS**

<table>
<thead>
<tr>
<th>Section</th>
<th>Subsections</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.01.00</td>
<td>General</td>
<td>90</td>
</tr>
<tr>
<td>90.10.00</td>
<td>The Role of the Monitor</td>
<td>00</td>
</tr>
<tr>
<td>90.10.01</td>
<td>General</td>
<td>00</td>
</tr>
<tr>
<td>90.10.10</td>
<td>The Effect of the Monitor on Performance</td>
<td>01</td>
</tr>
<tr>
<td>90.20.00</td>
<td>The Role of the Programmer</td>
<td>00</td>
</tr>
<tr>
<td>90.20.10</td>
<td>Planning for Performance</td>
<td>00</td>
</tr>
<tr>
<td>90.20.20</td>
<td>Organizing for Performance -- How to Use LOCALs</td>
<td>01</td>
</tr>
<tr>
<td>90.20.30</td>
<td>Programming for Performance</td>
<td>00</td>
</tr>
<tr>
<td>90.30.00</td>
<td>Reducing Core Storage Requirements</td>
<td>00</td>
</tr>
<tr>
<td>90.30.01</td>
<td>Programming Techniques to Increase Speed</td>
<td>00</td>
</tr>
<tr>
<td>90.30.10</td>
<td>The FIND Statement</td>
<td>00</td>
</tr>
<tr>
<td>90.30.20</td>
<td>Productive Time That Can be Improved by Hardware Changes</td>
<td>00</td>
</tr>
<tr>
<td>90.30.30</td>
<td>Nonproductive Time That Can be Reduced by Hardware Changes</td>
<td>00</td>
</tr>
<tr>
<td>90.40.00</td>
<td>Some Case Studies of Performance Improvements</td>
<td>00</td>
</tr>
<tr>
<td>90.40.01</td>
<td>General</td>
<td>00</td>
</tr>
<tr>
<td>90.40.10</td>
<td>Case I</td>
<td>00</td>
</tr>
<tr>
<td>90.40.20</td>
<td>Case II</td>
<td>00</td>
</tr>
<tr>
<td>90.40.30</td>
<td>Case III</td>
<td>00</td>
</tr>
<tr>
<td>90.40.40</td>
<td>Summary</td>
<td>00</td>
</tr>
</tbody>
</table>
GENERAL

This section covers many items of interest to all 1130 users:
• How to conserve core storage
• How to increase the running speed of a program
• How to segment programs
• The proper (and improper) use of LOCAL and SOCAL subroutines, etc.

The general theme of this chapter, is, however, how to improve your system, or, how to increase system performance.

The performance of your programs should be one of the major considerations of your programmer. Unfortunately, however, performance is all to often forgotten in the drive to produce a working program. The programmer, usually working against a deadline, devotes all his energy and ingenuity to the TEST/DEBUG/CORRECT/RETEST cycle, finally producing an error-free program with no time to spare, and with little thought given to efficiency.

Remember, however, that this program now enters production status, to be run weekly, or possibly daily, where its performance may greatly affect the overall operation of the 1130 system.

Program performance is affected by three factors, each of which will be discussed in more detail:
• The Monitor, or software
• The programmer
• The system itself, or 1130 hardware
THE ROLE OF THE MONITOR

General

The 1130 Monitor system has an outstanding feature, known as the "system overlay scheme", designed to assist you in fitting your programs into core storage.

This scheme is covered in some detail in Section 65, under "SOCALs".

Recapping that section briefly, the Core Load Builder, which is given the task of building a core load, or ready-to-execute package, also is given the task of resolving the problem of more program than core storage (if this problem arises).

Typically, many blocks of programming are competing for core storage: your programs, your subprograms, the IBM subprograms, and the Monitor control package itself. All must be in core storage when required.

As a first step, the CLB attempts to fit the entire package into core storage simultaneously. If that does not fit, the CLB splits the IBM subprograms into four groups:

- Group 0: Basic
- Overlay 1: Arithmetic (add, subtract, multiply, etc.)
- Overlay 2: Non-disk Input/Output (cards, printer, etc.)
- Overlay 3: Disk Input/Output

As step 2, the CLB determines whether the package will fit in core if Overlay 1 and Overlay 2 share the same area in core storage (the SOCAL area). The SOCAL area must be large enough to contain Overlay 3 plus the larger of Overlays 1 and 2.

If this does not provide enough room, step 3 is taken. Here, all three overlays (1, 2, and 3) will share the same area, which must now be as large as the largest overlay.

Step 4, taken if step 3 does not work, consists of a message informing you that this program is too large to fit in core storage.

To illustrate this graphically, Figure 90.1 shows the layout of the SOCA$Ls required by a "typical" commercial job. This "typical" program:

- Is written in FORTRAN.
- Adds, subtracts, multiplies, and divides.
- Uses the 1132 Printer, the 1442 Card Read Punch, and the console typewriter (but not the keyboard).
- Contains at least one PAUSE, STOP, and CALL DATSW statement.
- Contains disk READ, WRITE, and FIND statements.

If you punch an L in column 14 of the // XEQ card, the CLB will print a core storage map of your program and all its subprograms, indicating which are SOCAL or LOCAL, and what overlay level is in effect.
Figure 90.1.
The Effect of the Monitor on Performance

To return to the main subject of this chapter, you may ask, "How does all this affect performance?"
To answer this, we can construct a flowchart of a "typical" commercial job. Let us say it is basically of the type:
1. Read a card.
2. Taking a key item number from the card, look up its approximate disk location in an index table (indexed sequential organization).
3. Read a disk record.
4. Determine whether it is the right disk record. If it is, continue; if not, decrease the record number by 1 and go back to step 3.
5. Do some calculations based on the data obtained from the disk and from the card.
6. Write an updated disk record.
7. Print a line of answers on the 1132 Printer.
8. Do some arithmetic (reset indicators, clear totals, etc.) and go back to step 1.

For the purposes of this analysis you may ignore routines that are executed only once (initialization, final totals) or infrequently (error messages, etc.). Figure 90.2 shows this job in the form of a rough flowchart.

If this program is of a size that requires no overlays, it will run at some base speed or throughput rate. If its size is such that it must run at SOCAL level 1, Overlay 1 (Arithmetic) and Overlay 2 (Non-disk I/O) must be read from the disk whenever required. Figure 90.3 shows when these overlays would be required. This will lengthen the base running time.

Each pass will require four overlays and two disk arm moves. The arm moves are required because the disk data file and the SOCAL overlays are on different areas of the same disk. The time required for these arm movements varies, depending on several factors, but it may be considerable. A good average might be about 250 milliseconds or 1/4 second.

If the program must run at Overlay level 2, the picture changes considerably, as seen by Figure 90.4. If it hits the correct disk record on the first try, it will require seven overlays and four disk arm moves. For each additional disk read looking for the correct record, add two overlays and two arm moves. Running time will be further lengthened.

Figure 90.2. "Typical" commercial job -- rough flowchart
Figure 90.3. Overlays and disk arm movements required at SOCAL level 1

Figure 90.4. Overlays and disk arm movements required at SOCAL level 2
Figure 90.5 summarizes the overlay pattern as it varies with the disk search.

You can see the Overlay level 1 will not hurt performance too much. If each arm movement takes about 1/4 second, the processing time per card might jump from 8 to 8 1/2 seconds. Overlay level 2, on the other hand, may cause this program to run significantly more slowly. A typical indexed sequential file might require 15 disk reads to find the correct record. This would increase the time per card from 8 seconds to 16 seconds, or half the throughput rate. This could become even worse if the data file being searched were large or distant from WS, since the SOCAL area would be proportionately further away from the file area.

The overlay time itself may be ignored, since it is quite small compared with the disk arm movement time.

This example illustrates two principles:

1. The disk data file must be organized so that items may be found quickly (see Section 85).
2. For programs involving disk search techniques, as does the example, you should try diligently to avoid SOCAL Overlay level 2.

The difference between level 2 and level 1 is either 620 words (READ and WRITE disk) or 700 words (READ, WRITE, and FIND disk), but this does not mean that you must cut 620 or 700 words from your program to drop from level 2 to level 1.

The CLB will use level 2 if the program is too big for level 1. (It may be one word too big or 700 words too big.) Every word you can cut from the size of the program increases the probability that the program will fit at level 1 rather than level 2.

<table>
<thead>
<tr>
<th>Number of disk READs to find the desired record</th>
<th>Overlay level 0</th>
<th></th>
<th>Overlay level 1</th>
<th></th>
<th>Overlay level 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Overlays</td>
<td>Arm moves</td>
<td>Overlays</td>
<td>Arm moves</td>
<td>Overlays</td>
<td>Arm moves</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>25</td>
<td>22</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>35</td>
<td>32</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>45</td>
<td>42</td>
</tr>
</tbody>
</table>

Figure 90.5. Single-drive 1130 system

2. For this reason, you should strive to keep your programs as small as possible. Several means of doing this are discussed in the next subsection. Also, Section 70 gives many FORTRAN core saving tips.

Note that the above analysis applies to single disk drive 1130 systems; the addition of a second disk drive would eliminate all the overlay-caused arm movements — assuming of course, that you have placed your data file on one disk and Working Storage on another.
THE ROLE OF THE PROGRAMMER

In reading the preceding subsection, you may have got the idea that the 1130 Monitor has the major effect on the performance of your programs and that you do not enter the picture unless the "system overlay scheme" fails to squeeze your program into core storage.

Nothing could be further from the truth. The program the CLB was manipulating was, all, planned, organized, and programmed by you, not by the CLB.

Any one of these three functions, if not properly done, can force the CLB into building an inefficient package -- one that may take five or ten times longer in execution than a similar (but better planned) program doing the same job.

As mentioned earlier in this section there are many things you can do to avoid such inefficiencies; most of them are easy to understand, remember, and implement.
Planning for Performance

The major factor affecting program performance is core storage and how it is used. Therefore, you should try to avoid core storage difficulties by planning for reasonably sized program packages. It may seem quite efficient to have the entire payroll processed by one comprehensive program, but, overall, it would probably turn out to be quite inefficient. Because it would be a very large program, it would probably involve many overlays and could run for eight hours, whereas four smaller programs might take only five or six hours.

Section 60 contains many hints on how you may write small, modular programs. Besides helping to gain performance, modular programs have many other advantages over large, all-inclusive ones (they are easier to test, tend to keep errors from spreading, etc.).
After its scope has been determined, a program should be organized into logical blocks that lend themselves to efficient segmentation. You should organize your program expecting to have problems concerning core storage. If you do not have problems, very little time is lost. If you do, as is typical in most cases, you are in a position to create your own overlay scheme, if that of the loader will degrade the performance of your program.

As you have seen, in Section 65, the Monitor gives you two overlay or segmentation methods: LOCAL subprograms and program LINKs. These two overlay schemes are entirely planned and executed by you, in contrast to the Core Load Builder's automatic SOCALs.

The three interact in one important way: If you can conserve enough core with LOCALs and LINKs, the CLB will not have to resort to SOCALs. As you saw earlier, SOCAL Overlay level 2 can seriously degrade the performance of some programs, particularly those that search a disk data file looking for a certain key (man number, part number, etc.).

If you have a program such as this running at a comparatively slow rate, you should investigate it closely; if the program is using level 2 overlays, you should make a determined effort to reduce its size enough to allow CLB to use level 1. (To find out which overlay level is in use, execute the program with an L punched in column 14 of the \//XEQ card.)

Figure 90.6 shows the same program used earlier as an example. To it has been added:

- **INIT** The Initialization routine
- **WRAP** The wrap-up routine (grand totals)

and three exception subroutines:

- **BADCD** "Bad input card" message
- **NOHIT** "No such item on disk" message
- **NEWPG** Page heading routine

In addition, the following have been made into subroutines:

- **READC** Read card
- **CALC1** Calculations Part 1
- **CALC2** Calculations Part 2
- **CALC3** Calculations Part 3
- **MISC** Miscellaneous arithmetic

How should you go about reducing the size of this program? Many programmers, irked at the fact that their program does not fit in core storage, take an "I'll show 'em" attitude and make all subroutines LOCAL. This probably will eliminate
the need for Overlay level 2, but is a rather extreme case of over-reacting to a problem. Figure 90.7 shows the way in which this program would run, if all seven subroutines were LOCAL and Overlay level 1 were used. Each card processing cycle would involve 11 overlays (7 LOCALs, 4 SOCALs) and 4 arm movements (these figures are not dependent on the number of times the disk must be read before finding the desired item).

Reviewing Figure 90.7, it seems that you are somewhat better off than if you had used Overlay level 2, but you still require an excessive number of overlays and arm movements.

It would have been far more prudent to LOCALize only BADCD, NOHIT, and NEWPG, three subroutines that are only used occasionally. This would reduce your LOCAL overlays drastically and might save enough core storage for Overlay level 1 to be used.

Another technique that would reduce the size of this program is the use of LINKs. The blocks called INIT and WRAP could easily be separated from the main program, and made into what can be called "one-shot LINKs". This might save enough core storage to eliminate the need for LOCALs and SOCAL level 2 altogether.

Another LINK is possible here -- a type you might call a "repetitive LINK". Suppose you split the main program into:

**PART1**
- a. Read card
- b. Look up key in index
- c. CALL LINK (PART2)

**PART2**
- a. Read disk
- b. Check if correct record found
- c. Calculations
- d. Write new disk record
- e. CALL LINK (PART3)

**PART3**
- a. Print results
- b. Miscellaneous arithmetic
- c. CALL LINK (PART1) if not finished
  CALL LINK (WRAP) if finished

This arrangement is particularly good, for several reasons:
- It cuts the original program into five pieces (two small and three large).
- It isolates the I/O into separate LINKs -- for example:
  - PART1 uses neither the disk nor the printer.
  - PART2 uses neither card nor printer.
  - PART3 uses neither card nor disk.

This reduces the sizes of these LINKs substantially
- It probably eliminates the need for SOCALs and LOCALs altogether.

Figure 90.7.
To summarize, a typical program has been segmented in several different ways, and the probable effect of each way on performance has been discussed. The purpose has not been to illustrate that LINKs are better than LOCALs, or that LOCALs are better than SOCALs, or any other hard and fast rule. The purpose has been to illustrate that the options must be chosen wisely, not blindly. The easiest way, letting the CLB do it with SOCALs, may or may not be the best in terms of performance. The next easiest way -- LOCALs -- may or may not be best. The only way to determine which is best is to draw a flowchart of the type shown and to tailor the overlay option to the program.

You can generalize somewhat, with some common-sense do's and don'ts:

1. DON'T worry about the performance of a program that runs for only a few minutes, or that is used only occasionally. Concentrate your efforts on the long-running, everyday jobs.

2. DON'T place an overlay, or cause one to be placed, within a loop that reads from the disk. For example, take the problem discussed above, where you have a loop of the type:
   
   Read disk record  
   Compare disk key to sought-for key  
   If not equal, repeat

   SOCAL level 2 will overlay the Disk I/O package (required for the Disk READ) and the Arithmetic package (required for the subtraction within the IF statement parentheses). Furthermore, the disk READ command requires the disk arm to move away from the SOCAL disk area. This repetitive disk arm movement may have a disastrous effect on the running time of the program.

   If you place a LOCAL subroutine within this loop, it will have the same effect as if the CLB had included a SOCAL.

3. DON'T LOCALize subprograms that are always used, unless it is absolutely necessary to get the program into core storage. DO LOCALize subprograms that are the exception rather than the rule (error messages, new page headings, initialization, final totals, unusual payroll deductions, etc.).

4. DO minimize the amount of coding between DISK I/O commands. This, in turn, will minimize the chance of an overlay (SOCAL or LOCAL) which will require that the disk arm move from the data area to the overlay area and back again.

5. Also, DON'T LOCALize a subprogram that is called between two disk statements. For example, suppose a program has the following sequence:
   
   DISK I/O  
   CALL SUB  
   DISK I/O

   In this case SUB should not be made LOCAL, since it will force excessive disk arm movement.

6. DO plan for problems with performance -- either a program too large for core or a program that does not run as fast as it might. Keep the scope (and therefore the size) of each program modest; program as a series of LINKs; design the exception routines as subprograms; etc.
Programming for Performance

You have seen in the preceding examples that system performance is very closely related to the size of a program. In general, the larger the program, the more slowly it will run. This degradation is not evidenced in a gradual way; because of the SOCAL and LOCAL system, it will show up in sudden jumps or drops in throughput rates. Suppose you have an 1131 Model 2B (8K) and the familiar "typical" program. With no overlays you have about 4500 words for your program; with Overlay 1, about 4920 words; with Overlay 2, about 5620 words. Assuming these figures to be exact, this means that:

<table>
<thead>
<tr>
<th>If your program size is</th>
<th>It will:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 -- 4500 words</td>
<td>Fit with no overlays</td>
</tr>
<tr>
<td>4501 -- 4920 words</td>
<td>Require Overlay 1</td>
</tr>
<tr>
<td>4921 -- 5620 words</td>
<td>Require Overlay 2</td>
</tr>
<tr>
<td>5621 words or more</td>
<td>Not fit in core storage</td>
</tr>
</tbody>
</table>

If you add 1 word to a 4920-word program, it will suddenly require Overlay 2 and may take twice as long to execute. (It may also take no longer than before -- this depends on the program.)

Conversely, if you have a program at level 2, it may take anywhere from one word to 700 words to make it drop to level 1. If it was 4921 before, it will take only one word; if it was 5620, it will take 720 words.

Reducing Core Storage Requirements

To make a long story short, every word counts. You should always keep this fact in mind and strive to write efficient programs. Section 70 gives many core saving tips; Section 65 also gives some ideas for improving the SOCAL system. Repeating the FORTRAN tips (the details are given in Section 70, 50, 20):

1. Use the DATA statement wherever possible.
2. Keep FORMAT statements compact.
3. Take square roots and raise numbers to powers in the most efficient manner.
4. Code efficient I/O statements.
5. Avoid long subroutine argument lists.
6. Don’t include unneeded I/O devices on the *IOCS card.
7. Avoid arithmetic with constant subscripts.
8. Remove the TRACE from production status programs. The trace package requires about 140 words of core storage. In addition, it requires that Data Switch 15 be interrogated every time you "execute" an equal sign, IF statement, or computed GO TO. This requires 150 to 200 microseconds each time; some programs may do this tens of thousands of times in the course of one run.
Programming Techniques to Increase Speed

Just as reduced program size can improve performance, so can several programming techniques. All involve utilizing the overlapped I/O capability of the 1130. The hardware of the 1130 allows for the overlapping of almost all I/O devices; however, the programming system used determines which units can actually be made to run concurrently with other units, or with the central processor. (See Figure 90.8.)

Overlapping means that you can:
1. Tell the device what it is to do.
2. Start it going (printing, punching, etc.).
3. Then continue with other processing before the device has actually finished what it has started.

This section covers those units that can be overlapped by standard FORTRAN. The use of the overlapped I/O feature of the Commercial Subroutine Package is discussed in Section 70.

<table>
<thead>
<tr>
<th>Unit</th>
<th>FORTRAN</th>
<th>FORTRAN with Commercial Subroutine Package</th>
<th>Assembler</th>
</tr>
</thead>
<tbody>
<tr>
<td>1442-6 or -7 Reader</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1442-6 or -7 Punch</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1442-5 Punch</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Console Typewriter</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Console Keyboard</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1132 Printer</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1403 Printer</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Disk – Arm Movement (FIND)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>– Reading</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>– Writing</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>2501 Reader</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1627 Plotter</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1134 Paper Tape Reader</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1055 Paper Tape Punch</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Figure 90.8. Programming systems permitting overlapped operations

The FIND Statement. Because it is an optional feature of FORTRAN, some programmers are unaware of, and/or neglect, the use of the FIND statement. However, in many disk-oriented programs it can increase performance significantly. It can be added to any program quite easily and is simple to use.

Suppose your program calls for a disk read from record NR of file 6:

READ (6'NR) DATA

The disk subroutine will automatically compute where that record resides, move the disk arm to the proper position, and read the data. As mentioned many times earlier, the second job, the movement of the disk arm, may take much longer than the other two functions.

The FIND statement

FIND (6'NR)

ahead of the READ (or WRITE) will cause the subroutine to compute the location of record NR, start the disk arm moving to that location, and then continue processing other FORTRAN statements.

The secret of the FIND statement is self-evident: it should be placed as far in advance of the actual READ or WRITE statement as possible. In this way you can get the arm moving, overlapping its movement or "seek" time with computations, printing, etc.

Let us take a portion of a FORTRAN program that looks like this:

FIND (6'NR)

... other FORTRAN coding

READ (6'NR) DATA

Suppose it takes 700 milliseconds to move the disk arm to record NR from where it happens to be now. Suppose also, that the "other FORTRAN coding" shown takes 300 milliseconds. Without the overlapping gained by the FIND statement, the total time would be 700+300 or 1000 milliseconds. With the FIND statement, the total time would drop to 700 milliseconds, since the 300 milliseconds is "buried" within the 700 milliseconds seek time. Figure 90.9 illustrates this graphically. This might amount to only 30 or 30 minutes a day, but it is so easy to implement that it is certainly worth the trouble of punching a few FIND cards.

If you are using LOCALs, and/or the CLB has included SOCALs, the FIND statement will not be executed. The Monitor will take care of this automatically. The reason is obvious: if you FIND a
record then call a LOCAL or SOCAL subprogram, the entire purpose of the FIND will have been negated, and you will wind up increasing disk seek time rather than decreasing it. If you know you will have LOCALs or SOCALs, you may want to remove all the FIND statements from your program, eliminating the SDFND subroutine, which is approximately 80 words long.

Without the FIND statement:

```
Without the FIND statement:

READ

<table>
<thead>
<tr>
<th>Other Coding</th>
<th>Compute Location of Record</th>
<th>Arm Movement</th>
<th>Read Record</th>
<th>Continue</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300 msec</td>
<td>X</td>
<td>700 msec</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

Total time = 700 + 300 + X + Y
X is small compared with the others.
```

With the FIND statement:

```
With the FIND statement:

READ

FIND

<table>
<thead>
<tr>
<th>Compute Location of Record</th>
<th>Other Coding</th>
<th>Arm Movement</th>
<th>Compute Location of Record</th>
<th>Read Record</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>300 msec</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total time = 700 + X + X + Y
X is small compared with the others.
```

Figure 90.9.
THE ROLE OF THE 1130 HARDWARE

General

The last component in the user/hardware/software trio is the 1130 hardware itself. Because this section is concerned primarily with increasing performance, the discussion will concentrate on the ways you can improve throughput by the use of alternative hardware configurations.

The first step is the separation of run time into four basic elements:

1. Productive time that cannot be improved by hardware changes
2. Productive time that can be improved by hardware changes
3. Nonproductive time that cannot be improved by hardware changes
4. Nonproductive time that can be improved by hardware changes

The third is included only for completeness; using the definitions, there are no meaningful items to discuss in this area.

Productive time is the time that the 1130 occupies itself doing something you want it to do. Nonproductive time applies to activities that may be necessary, but that are unproductive from your point of view. Some examples of the latter are disk seeks, reading LOCAL and SOCAL overlays, etc.
Productive Time That Cannot Be Improved by Hardware Changes

Some of the 1130 system components are available in only one model; therefore, it is impossible to increase performance by changing them. The typewriter, the console keyboard, the paper tape reader, and the paper tape punch are four such devices. In addition, the reading/writing speed of the disk is constant, which means that the reading/writing of your data records cannot be speeded up through hardware changes. However, because more disk drives may be added, certain other times relative to the disk (seeks, reading of overlays) may be reduced; they are therefore covered in a later section.
Productive Time That Can Be Improved by Hardware Changes

There are five elements of productive time that can be improved by changing the model or speed of an 1130 component. That is, you can:

- Reduce plotting time by switching to a faster plotter
- Reduce card reading time by obtaining a faster card reader
- Reduce card punching time by obtaining a faster card punch
- Reduce printing time by obtaining a faster printer
- Reduce computing time by changing to a faster CPU

Plotting

This is a somewhat special case; two plotting speeds are available, but they are tied to carriage sizes. The 1627 Model 1, with an 11-inch carriage, is twice as fast as the Model 2, which has a 29 1/2-inch carriage. However, most users have chosen one model or the other on the basis of carriage size, rather than speed, and are not in a position to change models just to increase speeds.

Card Reading

There are four different card readers that may be attached to the 1130 system, each with a different card-read time:

<table>
<thead>
<tr>
<th>Reader</th>
<th>Milliseconds per card (approx.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1442 Model 6</td>
<td>200</td>
</tr>
<tr>
<td>1442 Model 7</td>
<td>150</td>
</tr>
<tr>
<td>2501 Model A1</td>
<td>100</td>
</tr>
<tr>
<td>2501 Model A2</td>
<td>60</td>
</tr>
</tbody>
</table>

If your programs use standard FORTRAN, none of the specified card read time will be overlapped with any other activity.

If you have a 1442-6 on your 1130, for example, the time to read ten cards will be 10 X 200 or 2000 milliseconds. This is in addition to whatever manipulation must be performed on the data on those cards. In a FORTRAN program, the system must, at the very least, convert the Hollerith card codes to EBCDIC, break that down according to the specified FORMAT statement, and, finally, place the resulting data in the proper core location.

The rated speed of the 1442-6 is 300 cards per minute, but this assumes that the 1130 reads a card every 200 milliseconds. It is true that the reading of each card will take 200 milliseconds, but the system may not read a card every 200 milliseconds. If the intervening processing takes 100 milliseconds, it will read one card every 300 milliseconds, yielding a speed of 200 cards per minute.

You see, then, that rated I/O device speeds are difficult to use when evaluating potential system improvements. You must compare alternatives on the basis of the time per card that the CPU is prevented from doing something else.

Suppose you have a 1442-6, and you time one of your representative jobs. It reads 2100 cards, and runs for ten minutes (600,000 milliseconds). You know, from the timing table, that the 1130 must have spent 2100 X 200 or 420,000 milliseconds reading cards, and 600,000-420,000 or 180,000 milliseconds doing something else.

If you changed to a 1442-7, the card read time would drop to 2100 X 150 or 315,000 milliseconds, the "something else" would remain at 180,000 milliseconds, and the total run time would drop from 600,000 milliseconds to 495,000 milliseconds, or from ten minutes to about 8 1/4 minutes.
The 2501, because of its clutch arrangement, requires a special analysis. The 2501-A1, the 600-card-per-minute reader, will read at fixed speeds of

- 600 cpm (100 millisec)
- 300 cpm (200 millisec)
- 200 cpm (300 millisec)
- 150 cpm (400 millisec)
- 120 cpm (500 millisec)
- 100 cpm (600 millisec)

etc.

and the 2501-A2, the 1000-card-per-minute reader, will read at fixed speeds of

- 1000 cpm (60 millisec)
- 500 cpm (120 millisec)
- 333 cpm (180 millisec)
- 250 cpm (240 millisec)
- 200 cpm (300 millisec)
- 166 cpm (360 millisec)

etc.

To calculate the expected improvement in timing due to a 2501-A1, we must, as before, substitute 100 milliseconds for the 200 milliseconds (1442-6), to get 2100 x 100 or 210,000 milliseconds read time, add the 180,000 milliseconds other time, obtaining 390,000 milliseconds or 15 minutes. Dividing this into the number of cards read (3000), we find that this yields a rate of 323 cards per minute.

However, the clutch arrangement of the 2501-A1 will not allow it to run at 323 cards per minute, so the next lower speed (300 cpm) must be assumed. 2100 cards at 300 cpm yields a total time of seven minutes.

A similar analysis for the 2501-A2 gives a theoretical speed of 412 cpm, but, choosing the next lower speed, 333 cpm, the total run time is calculated as 6.6 minutes.

Card Punching

Three different card punches are available for use on the 1130 system; all three operate in the same mode, punching one column at a time.

<table>
<thead>
<tr>
<th>Card Punch</th>
<th>Milliseconds per Card Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>1442 Model 6</td>
<td>12.5 plus 12.5 per column</td>
</tr>
<tr>
<td>1442 Model 7</td>
<td>6.5 plus 6.5 per column</td>
</tr>
<tr>
<td>1442 Model 5</td>
<td>6.5 plus 6.5 per column</td>
</tr>
</tbody>
</table>

Models 6 and 7 both read and punch; Model 5 only punches.

The overall speed is determined by the last column punched, rather than the number of columns punched. If you skip the first 20 columns and punch into the 21st, you have punched or spaced 21 columns and the time for that number will apply. Figure 90.10 gives the punching time for the three models, as they vary with the last column punched.

To continue the previous example, suppose that of the 2100 cards read, the program punched into...
the first 20 columns of 500 of them. For the 1442-6, the breakdown now becomes:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read 2100 cards</td>
<td>420,000</td>
</tr>
<tr>
<td>Punch 20 columns, 500 cards</td>
<td>131,250</td>
</tr>
<tr>
<td>Something else</td>
<td>48,750</td>
</tr>
<tr>
<td>Total</td>
<td>600,000</td>
</tr>
</tbody>
</table>

With the 1442-7, it becomes:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read 2100 cards</td>
<td>210,000</td>
</tr>
<tr>
<td>Punch 20 col. 500 cards</td>
<td>68,250</td>
</tr>
<tr>
<td>Something else</td>
<td>48,750</td>
</tr>
<tr>
<td>Total</td>
<td>327,000</td>
</tr>
</tbody>
</table>

or 5.5 minutes

Note that the times shown apply only to the time actually spent punching. If the card being punched was previously read, the punch time may be simply added to the total. If the card being punched was not previously read, you must add 200 or 150 milliseconds of read time per card to allow for the feeding of cards past the read station, even though they were not read. This will always be the case with the 1442-5, which cannot read cards.

### Printing

Three different line printers may be attached to the 1130 system, each having different print and skip times:

<table>
<thead>
<tr>
<th>Printer</th>
<th>Approximate Time in Milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Print 1 Line</td>
</tr>
<tr>
<td>1132</td>
<td>750</td>
</tr>
<tr>
<td>1403 Model 6</td>
<td>175</td>
</tr>
<tr>
<td>or</td>
<td>(3.6-microsecond CPU)</td>
</tr>
<tr>
<td>1403 Model 7</td>
<td>100</td>
</tr>
<tr>
<td>or</td>
<td>(2.2-microsecond CPU)</td>
</tr>
</tbody>
</table>

To illustrate the improvement possible in this area, let us take an example similar to the last one. Suppose you have a program that is essentially a card listing job. In ten minutes it reads 600 cards, prints 600 lines, and skips 100 lines. This can be broken down as follows:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read card (1442-6)</td>
<td>120,000</td>
</tr>
<tr>
<td>Print (1132) 600 @ 750</td>
<td>450,000</td>
</tr>
<tr>
<td>Skip 100 @ 16</td>
<td>1,600</td>
</tr>
<tr>
<td>&quot;Everything else&quot;</td>
<td>28,400</td>
</tr>
<tr>
<td>Total</td>
<td>600,000 (10 minutes)</td>
</tr>
</tbody>
</table>

If you replace the 1132 with a 1403-6, your print and skip times drop:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print (600 @ 175)</td>
<td>105,000</td>
</tr>
<tr>
<td>Skip 100 @ 5</td>
<td>500</td>
</tr>
</tbody>
</table>

Added to the card read time and the "everything else" time, which remains the same, this results in a total time of 253,900 milliseconds, or about 4 1/4 minutes, as opposed to 10 minutes.

Note that despite this dramatic increase in throughput, the 1403 is printing at only 141 (600/4.25) lines per minute, far below its rated speed of 340. The 1132 was also below its rated speed of 80 lpm, since it printed 600 lines in ten minutes, or 60 lpm.

This shows again that rated speeds of cards per minute, lines per minute, etc, cannot be used when investigating alternate approaches to improving throughput. The only usable figure is the length of time the CPU is tied up—- that is, prevented from doing something else.

This example has assumed a 3.6-microsecond CPU; if the 1130 were a Model C (2.2 microseconds), a 1403 time of 100 milliseconds would be used. The overall time would drop to 3.5 minutes, for a speed of 172 lpm.

In all cases of 1403 timing investigations, you must calculate the resulting lines per minute to make sure that it does not exceed the rated speed of the
printer. For example, an analysis that indicates a 1403 speed of 450 lpm must be modified if the printer considered is a 1403-6, which cannot exceed 340 lpm.

The 750-millisecond time for the 1132 is based on standard FORTRAN, which is not overlapped.

The 176 (or 100) millisecond time for the 1403 consists mainly of the conversion from EBCDIC to 1403 code - the 1403 itself is buffered, and the time required to fill the buffer is quite small. The 176 milliseconds drops to 100 on a 2.2 microsecond CPU because of the faster CPU. See the next subsection.

Computing

The 1131 Central Processing Unit is available with one of two basic cycle times: 3.6 microseconds (Models 1 and 2) or 2.2 microseconds (Model 3). In more basic terms, the Model 3 will compute in 0.61 the time of the Model 1 or 2.

However, in this area it is not quite as easy to calculate the improvement to be expected from the faster CPU. The problem is that you often don't know how much time you were computing before (with a 3.6-microsecond CPU), in which case you cannot possibly tell what effect the 2.2-microsecond CPU will have.

Let us review the previous example: 1442-6 and 1132; ten minutes run time, read 600 cards, print 600 lines, skip 100 lines. The times in milliseconds, were:

<table>
<thead>
<tr>
<th>Activity</th>
<th>Time in Milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Card read</td>
<td>120,000</td>
</tr>
<tr>
<td>Print and skip</td>
<td>451,600</td>
</tr>
<tr>
<td>&quot;Everything else&quot;</td>
<td>28,400</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>600,000</strong></td>
</tr>
</tbody>
</table>

The only way you determined the 28,400 milliseconds of "everything else" was by subtracting one known value (I/O times) from another known value (total run time).

If you know that all 28,400 milliseconds were spent in computing, you can calculate that the 2.2-microsecond CPU will do the same amount of work in 61% of that time, or 17,300 milliseconds, a reduction of 1,100 milliseconds or 1.8 minutes.

If those 28,400 milliseconds had included any disk operations, you could not have made the above estimate, since you would have had no way to determine the split between disk activity and computing. Aside from a good estimate, which would be quite an achievement, the only way to evaluate the effect of a new CPU in this case would be to take your program to such an 1130, run it, and time it.
Nonproductive Time that Can Be Reduced by Hardware Changes

By definition, three items fall into this category:

1. DISK seek, to get from one data record to the next
2. DISK seek, to get from data area to overlay area, and vice versa
3. DISK read to read overlay

All three items are necessary, but unproductive as far as you are concerned. Note that item 1 is required whenever you are using data files, item 3 whenever you are using overlays (SOCALs, LOCALs, and/or LINKs), and item 2 whenever you have both overlays and data files.

The time requirements of all three are difficult to determine, so an exact analysis will not be attempted, as with the card readers, punches, etc.

There are two hardware changes that will reduce these times:

1. More core storage, which will probably eliminate overlays, and therefore items 2 and 3.
2. More disk drives, which will allow a redistribution of files and overlays, and reduce items 1 and 2.

Additional Core Storage

Aside from programmer convenience, the main advantage in adding more core storage is its probable effect on performance, or run time. If you can execute your programs without any overlays, they can be expected to run at some "top" speed, governed mainly by the amount of productive work you want done.

Additional Disk Drives

Unlike core storage, which will probably be augmented to improve performance, additional disk drives are likely to be considered primarily to increase capability—the capability to copy disks, the additional storage gained, etc. In many cases, however, the move from a single to a multiple disk 1130 system may be accompanied by a gain in throughput or performance. This will be true only if you plan your system so that the LOCAL/SOCAL overlays are on a cartridge other than the one on which the data files reside.

The location (cartridge ID number) of the data files is specified on the *FILES card. The LOCAL/SOCAL overlays are either (1) in Working Storage, if the program is executed immediately after compilation, or (2) with the mainline program (in UA or FX), if the program has been stored in core image format. If they are in Working Storage, the Monitor should be informed, with the JOB card, to use the Working Storage on a disk cartridge other than the data file cartridge. If they are with the mainline program (in UA or FX), you should make sure the core load is stored on a cartridge other than the data file cartridge.
SOME CASE STUDIES OF PERFORMANCE IMPROVEMENTS

General

This section is designed to present a general guide to the principles involved in improving performance. It also shows many of the techniques used to fit a large problem into core, stressing how to do so without adversely affecting performance.

In order to best illustrate these principles, three case studies, or sample problems, are shown in detail:
- Case I -- a commercial job, typical of a payroll-type application
- Case II -- a commercial job, typical of an accounting type application
- Case III -- a scientific or technical job, involving mostly computation, with little or no input/output

All examples are based on an 8K 1130 system, but the principles are the same for any size machine.
Case I

The first example uses a typical payroll-type application to show one approach to improving performance. It may not be the best approach, but it results in a set of programs that produce the desired result, fit in core storage, and operate at a near-maximum throughput rate.

A rough block diagram of this job, marked to show what action has been taken, is included with each step.
Step 1

The first time we are able to try to execute the program PAYRO we are informed that it does not fit in core storage, needing 388 (hexadecimal) or 904 words.

```
// XEQ PAYRO L 1
*FILESI*FILEN)
FILES ALLOCATION
 1 01A3 0001 7061 FILEN
22 0000 0001 7061 01A7
STORAGE ALLOCATION
R 40 07AD (HEX) ADDITIONAL CORE REQUIRED
R 43 01FC (HEX) ARITH/FUNC SOCIAL W/CNT
R 44 06EB (HEX) F/I/O SOCIAL W/CNT
R 45 02A2 (HEX) DISK F/I/O SOCIAL W/CNT
R 40 0388 (HEX) ADDITIONAL CORE REQUIRED
R 18 PAYRO LOADING HAS BEEN TERMINATED
```
Step 2

In order to test the program, we make all five subroutines LOCAL and find that it now fits in core, but requires SOCAL level 2. Running of the program is accompanied with quite a bit of disk arm movement, which slows it down considerably.

The subroutines are:
- **SUBW** -- Error message (hardly ever called)
- **SUBZ** -- New page headings (once every 25 employees)
- **SUBY1** -- FICA routine (almost always called)
- **SUBY2** -- Special deductions (one out of every six employees)
- **SUBY3** -- Savings Bond deduction (one out of every three employees)
Step 3

Studying the flowchart, we see that this program could be split into three smaller programs, or LINKS:

- PGMAB, which is made up of blocks A and B
- PGMX, which was block X
- MAIN, which is the main program

Executing with no LOCALs, we find that the program MAIN requires SOCAL level 2 to fit into core, and that it runs no faster than before.
Step 4

Making all five subroutines LOCAL again, we find this is just enough to eliminate SOCALS, but does not speed up the program, since SUBY1, the FICA routine, is called for almost every employee and causes the disk arm to be moved from the data file area back to the overlay area, and vice versa.
Step 5

Since SUBY1 as a LOCAL is slowing down the program, we must try to keep it in core storage at all times. However, the previous load map showed that there are only four words unused by the package, and SUBY1 is 400 words long. If we could free up 396 words, SUBY1 could be taken out of the LOCAL category, and the program would be speeded up.

(Realize, of course, that SUBY1 could easily be made non-LOCAL, but that SOCALs would then be required. The secret is to avoid both SOCALs and a LOCAL SUBY1).

Note also that SOCALs would cause the program to run even slower. Since the sequence of the program is a repetition of

a. I/O
b. DISK
c. ARITH, including SUBY1
d. DISK
e. ARITH
f. I/O

SOCAL level 1 will cause the disk arm to be moved between the data area and the overlay area between steps

a and b
b and c
c and d
d and e

while SUBY1 as a LOCAL will require such a movement only between steps

b and c
c and d

After considerable study, we decide that there is very little that can be done to further improve the performance of this program, unless, of course, we can reduce its size by 396-100 or 296 words (Flipper would no longer be required).

Because SUBY1 handles the FICA calculation, it will be called less and less as the year progresses, since more employees will attain a "paid up" status. (This won't be true, however, if your test for "paid-up" is done inside the subroutine! It should be made in the mainline program, otherwise SUBY1 will be called every time, whether the employee gets a deduction or not.)

Discussion of Case I

Here you have seen one way to fit this "typical" program into core, at little or no sacrifice in throughput. There may be other ways to do the same thing; there may be better ways.

Basically, common sense is used -- a step-by-step segmentation of the program, with each step having a greater effect on performance:

1. Make LOCALs out of those subroutines that are not always called.
2. Break the program into LINKs.
Case II

This program is of a basically different organization than Case I. It is typical of a job in which the input consists of a master card followed by a variable number of detail cards, with the sequence repeated many times. Some good examples of this type of job are billing, accounting, cost systems, etc.

Assume that this application is some type of project cost system, with a master card for each project, followed by a series of detail or change cards pertaining to that project. These detail cards may be due to labor or materials charges against the project or, in a few cases, an accounting department adjustment.
Step 1

After several tries, the program COST achieves a successful compilation, only to be met by the R40 and R18 messages shown. Even after SOCAL level 2 has been attempted, this program package exceeds core storage by 43E or 1086 words.

```
// XEQ COST L 1
*FILES(1:FILEN) FILES ALLOCATION
  1 01A3 0001 7061 FILEN
  22 0000 0001 7061 01A7
STORAGE ALLOCATION
  R 40 084C (HEX) ADDITIONAL CORE REQUIRED
  R 43 01E6 (HEX) ARITH/FUNC SOCAL WD CNT
  R 44 06E8 (HEX) FI/O, I/O SOCAL WD CNT
  R 45 02A2 (HEX) DISK FI/O SOCAL WD CNT
  R 40 043E (HEX) ADDITIONAL CORE REQUIRED
  R 18 COST LOADING HAS BEEN TERMINATED
```
Step 2

Observing the flowchart, we see that we are fortunate in having several subroutines that are seldom, if ever, called:

- BADCD, the illegal card message
- NEWPG, the skip to new page routine
- FINAL, the final total routine
- T, U, V, W, four routines involved in the processing of an accounting adjustment card (an unusual occurrence)

These seven subroutines are ideal LOCALs, and, executing COST in this mode, we get the load map shown. The program (at SOCAL level 2) runs, but quite slowly. Checking the flowchart, we see we have two blocks involving disk READs/WRITEs, D and H, bracketing blocks E, F, and G, which use both arithmetic and non-disk I/O functions. Obviously, this will cause continuous disk arm movement between the disk data file area and the overlay area.

The only way we can reduce this time-consuming function is to eliminate the need for overlays between the disk READ and WRITE.
Step 3

As mentioned in Step 2, we now realize that there is no real reason for blocks E, F, and G to be sandwiched between the disk READ and the disk WRITE.

Rearranging the program slightly, to make the sequence Z, E, F, G, D, H, we reexecute and find that the program runs substantially faster than before. There is still some disk arm movement, but it is not quite as frequent. Actually, as long as we have disk data files and overlays, there will be some disk arm movement. The goal is to reduce it, if it cannot be eliminated altogether.
In step 3, we have prevented overlays from occurring between disk READs/WRITEs. The next logical step is to eliminate overlays altogether, or, if that is impossible, limit overlays to LOCALs or SOCALs that are infrequently called.

Further study of the flowchart reveals that a master card is somewhat exceptional, even though every eighth card or so is a master card. Adding D, E, F, and G to the LOCAL list, we again execute and find that the program now runs even faster than before, with disk arm movement only when a master card is encountered. The load map shows that SOCALs are no longer required.
Discussion of Case II

Here, as in Case I, we take a similar series of common-sense steps to improve performance:

1. Make the exception subroutines LOCAL.

2. If that still requires SOCALs, consider separating the program into LINKs. In this case, this approach did not seem to be too effective.

3. Since SOCALs seem unavoidable, we try to rearrange our program steps to reduce their effect.
Case III

Here you have a technically oriented job, with a great deal of iterative or trial-and-error computation and very little input/output. The program reads a deck of ten cards, computes for quite some time, then prints a page of answers. On the basis of a similar program, you estimate that the computations should take about 15 minutes.
Step 1

Attempting to execute this program, TECH, for the first time, we are informed that it exceeds core storage by 2A0 or 528 words.

- // XEQ TECH L 1
  *FILES(1),*FILE*EN*)
- FILES ALLOCATION
  1  01A3  0001  7061  FILE
  22  0000  0001  7061  01A7
- STORAGE ALLOCATION
  R  40  08BC (HEX) ADDITIONAL CORE REQUIRD
  R  43  01C4 (HEX) ARITH/FUNC SOCAL WD CNT
  R  44  06E8 (HEX) FI/O x 1/O SOCAL WD CNT
  R  45  02A2 (HEX) DISK FI/O SOCAL WD CNT
  R  46  02A0 (HEX) ADDITIONAL CORE REQUIRD
- R 1b TECH LOADING HAS BEEN TERMINATED
Step 2

Noting that the program may be split into three separate programs or LINKs, we make some minor modifications and obtain:

- INPUT, made up of the first two blocks, A and B
- ANSWR, the printing of the results, formerly block K
- TECH1, the main program

Executing, we find that INPUT and ANSWR fit with room to spare, but TECH1 is still too large; however, it now exceeds core by only 8E or 142 words.
Step 3

Reexecuting TECH1 with all eight subroutines as LOCALs (L, M, N, P, Q, X, Y, Z), we learn from the load map that this strategy not only gets the program into core storage, but eliminates the need for SOCALs. It runs quite slowly, however, and takes nearly 60 minutes to go to completion, compared with the 15 minutes we expected. The sound of the disk arm moving gives us a clue to what is wrong: we have caused an overlay to be placed between the disk READ/WRITE commands. In this case the LOCAL subroutines L, P, and Q are the culprits.
A. Read input cards

B. Initialize

C. Compute:
   Call L
   Call M
   Call N

D. Write disk record

E. Compute:
   Call L
   Call P
   Call Q

F. Write disk record

G. Read disk

H. Compute
   Call X
   Call Y
   Call Z

I. Type message:
   "STEP NUMBER n"

J. Wrap-up computations

K. Print results, EXIT

Sizes of the Subroutines used:
- \#L: 100 words
- \#M: 300 words
- \#N: 300 words
- \#P: 400 words
- \#Q: 400 words
- \#X: 100 words
- \#Y: 300 words
- \#Z: 100 words

\* MAKE THESE LOCAL
Step 4

Leaving L, P, and Q off the LOCAL card, we again execute TECH1, but find that it runs even more slowly, since we now need SOCAL level 2 to fit into core storage.

At this point, you have a choice: accept the program as a one-hour job, or work on it further to speed it up. Since it is used quite often, you decide to give it one last check.
A. Read input cards

B. Initialize

C. Compute:
   - Call L
   - Call P
   - Call Q

D. Write disk record

E. Compute:
   - Call E
   - Call M
   - Call N

F. Write disk record

G. Read disk

H. Compute:
   - Call X
   - Call Y
   - Call Z

I. Type message: "STEP NUMBER n"

J. Wrap-up computations

K. Print results, EXIT

Sizes of the Subroutines used:

- L 100 words
- M 300 words
- N 300 words
- P 400 words
- Q 400 words
- K 100 words
- Y 300 words
- Z 100 words

*L MAKE THESE LOCAL*
Step 5

After some study, we notice that the typewritten message, block I, is the only non-disk input/output in the entire program. It looks innocent enough, but because of it, the entire Format Interpreter (SFIO) is required, plus the Typewriter routine (WRTYZ) and the typewriter code conversion routine (EBCTB). The total size of this package may be determined from the previous R44 message -- 514 (hexadecimal) or 1300 words.

Removing that message -- and the *IOCS (TYPEWRITER) Card! -- we recompile the program (calling it TECH2) and find, on execution, that it runs with no SOCALS or LOCALs.

It now executes to completion in 15 minutes, as we hoped, and the disk arm movement is reduced to an occasional "click" as it moves from one cylinder to the next in the data file area.

If a typewritten message is really needed, consider using the TYPER routine of CSP - it is quite small and does not use SFIO.
A. Read input cards

B. Initialize

C. Compute:
   Call L
   Call M
   Call N

D. Write disk record

E. Compute:
   Call L
   Call P
   Call Q

F. Write disk record

G. Read disk

H. Compute
   Call X
   Call Y
   Call Z

I. Type message: "TYPE NUMBER n"

J. Wrap-up computations

K. Print results, EXIT

Sizes of the Subroutines used:

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>100  words</td>
</tr>
<tr>
<td>M</td>
<td>300  words</td>
</tr>
<tr>
<td>N</td>
<td>300  words</td>
</tr>
<tr>
<td>P</td>
<td>400  words</td>
</tr>
<tr>
<td>Q</td>
<td>400  words</td>
</tr>
<tr>
<td>X</td>
<td>100  words</td>
</tr>
<tr>
<td>Y</td>
<td>300  words</td>
</tr>
<tr>
<td>Z</td>
<td>100  words</td>
</tr>
</tbody>
</table>

NO LOCALS REQUIRED!
Discussion of Case III

This type program, although quite different from the previous two cases, is analyzed in much the same way:

1. The main program is split into three LINKs: Input, Processing, and Output.

<table>
<thead>
<tr>
<th>No Disk Data Files</th>
<th>Program will run at some basic &quot;Top Speed&quot;.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small to Medium-Size Files Near WS (at the End of WS)</td>
<td>Program will run at some basic &quot;Top Speed&quot;, but probably not enough to be noticed.</td>
</tr>
<tr>
<td>Very Large Disk Data Files, or Small Files Deep Inside UA</td>
<td>Program will run at some basic &quot;Top Speed&quot;, but probably not enough to be noticed.</td>
</tr>
</tbody>
</table>

SOCAL's, LOCAL's and/or LINK's are used

- Overlays Limited to Seldom-Used Blocks
- Overlays Used Continuously, But Not In between Disk Statements
- Overlays Used in between Disk Read/Write Statements

2. Since all subprograms are called during each pass, we try to LOCALize only those that do not appear inside the main disk READ/WRITE loop.

3. With excessive overlays still required, we attack the main program and try to shorten it or eliminate some of the subroutines it uses.

Section Subsections Page

<table>
<thead>
<tr>
<th>90</th>
<th>40</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 90, 11.

Figure 90, 12.
Summary

To recapitulate the lessons learned in the preceding three case studies, performance depends on five major factors:

1. The size of the program. When writing any program, you should anticipate problems with core storage and performance. Plan programs of reasonable scope, and code them as a series of LINKs, if at all possible.

2. The subroutines required by the program. Realize that many seemingly innocent FORTRAN statements can cause sizable subroutines to be included in your core load. Some examples are PAUSE, STOP, FIND, division, use of the data switches, etc. FORTRAN control cards can have a similar effect -- for example, unnecessary *Iocs cards, the TRACE, etc.

3. The way the program is structured. When flowcharting and coding your programs, always keep in mind the location of the disk arm, so that you do not invite excessive arm movement between the overlay area and the data area. Place as little coding as possible between disk READ/WRITE loops so that the chance of an intervening overlay is reduced. Figure 90.11 shows the various combinations of data files and overlays.

Note that the location of the overlay has a great effect on performance. If you must move the disk arm from one area to the other, you can at least try to minimize the number of times it is required (or reduce the distance involved, by making data files compact).

4. The overlay scheme used. If your program is of such magnitude that some overlaying is required, you should have a good feel for how each works and how each can affect performance. Figure 90.11 shows that there is no differentiation made between LOCALs, SOCALS, and LINKs -- they are all overlays.

Note also that the number of times an overlay is required is not as important as the disk arm movement that may be necessary to get it. For this reason you should take particular care to avoid causing an overlay to be placed in between disk READ/WRITE statements.

LOCALs, because they are selected by the programmer, will often yield better performance than SOCALS, which are chosen by the CLB according to predetermined rules. However, if you select LOCALs without regard to their effect on performance, it is possible that they can slow down execution time even more than SOCALS.

5. The size and location of the data file. Since you are concerned with minimizing disk arm movement time, you should try to shorten the distance involved.

The overlay area is always at the end of the UA or at the beginning of WS, whichever way you prefer to look at it. The data files may be either:

- In the UA or FX, if you have put them there with the *STOREDATA card
- At the end of UA (beginning of WS), if you have not used a *STOREDATA or *FILES card

If you have a temporary file, in WS, your arm movement times will be minimized, since the files and the overlays are as close as they can be. If your file is in the UA, however, the picture may be quite different, depending on how "deep" it lies in the UA. If a DUMPLET shows that there is a great deal of distance between the file and the end of UA, you should consider moving the file. Figure 90.12 shows three possible situations.

The key to gaining good program performance is knowledge:

- Knowledge of the way in which the three overlays work
- Knowledge of the basic workflow of your program
Direct access storage device: 15.10.10
Disk (see "disk data file", "disk cartridge", etc.)
Disk arm movement time: 45.10.00
Disk cartridge: 15.20.40, 45.10.00, 80.10.00
   Checking of ID numbers: 55.30.00
   Format of material: 60.30.01
   ID numbers: 50.01.00
   Number required: 50.01.00
   Scratch: 50.01.00
   Space on: 60.20.10
Disk core image: 60.30.01, 80.30.20
   Disk core image format (see "disk data formats")
Disk data file: 15.10.01, 45.10.00
   Adding items: 85.10.10, 85.10.20
   Addition area: 85.10.10
   Backup: 15.10.60, 15.20.60
   Changes: 15.10.30
   Design: 20.30.10
   Duplicate copies: 15.20.10
   Hazards: 15.20.20
   Inquiry: 15.10.40
   Intentional modification: 15.20.20
   Jobs involving more than one: 15.10.20
   Organization: 85.10.01, 85.10.20
   Organization, choosing: 85.30.10
   Organization, direct: 85.10.30
   Organization, indexed sequential: 20.30.10, 75.20.10, 85.10.20
   Organization, partitioned direct: 85.10.30
   Organization, pure sequential: 85.10.10
   Organization, random: 20.30.10, 75.20.10, 85.10.30
   Organization, searching a pure sequential: 85.10.10
   Organization, sequential: 75.20.10
   Processing: 85.20.00
   Processing, random: 85.20.00
   Processing, sequential: 85.20.00
   Reorganization: 15.10.30, 85.10.10
   Safeguarding: 15.20.01
   Setup: 80.70.10
   Size: 15.10.50
   Space required: 80.40.00
Disk data formats: 60.30.01, 80.30.20
   Conversion: 60.30.20
Disk drives
   Logical: 50.01.00
   Physical: 50.01.00
   Disk file (see "disk data file")
   Disk management: 50.01.00
   Disk Monitor System: 50.01.00
   Version I: 70.20.10
   Version II: 65.10.00, 70.20.10
Disk storage (see "disk data file", "disk cartridge", etc.)
   Disk system format: 60.30.01
   Disk Utility Program: 50.01.00, 60.30.01
   DIV: 70.10.30
   Divisor: 70.10.30
   Documentation: 10.01.00, 15.20.70
   Current system: 20.01.00
   Old system: 40.20.00
   Standards: 25.10.00
   Document controls: 20.10.20
   Document register: 20.10.20
   DSP (see "disk system format")
   Dump: 60.30.20
   Dump a data file and reload: 60.30.20
   *DUMPLET: 90.30.40
   DUP (see "Disk Utility Program")
   Duplicate file: 15.10.60
   EDIT: 25.40.10, 70.30.00, 70.40.10, 70.40.20
   Editing: 25.40.10
   Input cards: 85.30.10
   EDIT mask: 70.40.20, 70.50.10
   Employee numbers: 85.10.30

EQUIVALENCE statement: 70.50.10
Error recovery sheet: 15.20.70
Errors: 15.20.20, 15.20.40, 15.20.50, 15.20.60, 15.20.70, 30.10.00,
   40.30.00
   Card punch: 30.01.00
   Program logic: 30.01.00
   Programmer clerical: 30.01.00
   Programmer procedural: 30.01.00
Exchanging sorting: 75.30.10
Execution time (see "running time, factors affecting")
Exponentiation: 70.50.20
Extended precision: 70.10.20, 80.50.00, 80.60.00
Fields: 10.10.00, 80.20.00
Field size: 20.30.10
File maintenance: 20.30.10 (see also "disk data file, organization")
   File organization: 20.30.10
   FILES: 80.20.00, 80.30.20, 80.70.10
   FILL: 70.10.30, 70.40.20
   FIND: 65.10.30, 70.50.20, 90.20.30
   Fixed area: 60.10.50, 60.20.20, 80.30.20, 80.70.10
   Fixed Location Equivalence Table: 60.10.50, 80.30.20
   Fixed point arithmetic: 70.10.20
   FLET (see "Fixed Location Equivalence Table")
   Flipper: 65.10.20, 65.10.30
   FLOAT: 70.40.10
   Floating boundary: 60.10.40
   Floating point arithmetic: 70.10.20
   Flowcharts: 10.01.00, 20.01.00, 25.10.00, 25.30.20, 30.40.00
   Format, A1: 70.40.10
   Format, A2: 70.40.10
   Formats, core storage required: 70.50.10
   Forms
   Design: 20.30.10, 20.40.20
   Preprinted: 45.40.00
   FORTRAN: 20.60.01
   Compiler: 50.01.00, 60.10.20
   Fractions: 70.10.20
   FUNCTION, arithmetic statement: 25.40.40
   FX (see "fixed area")
   GET: 70.30.00, 70.40.10
   GO TO, computed: 30.20.00
   Graphic output: 45.30.00, 45.35.00
   Half-adjust: 25.40.40, 70.40.10
   Hash total: 20.10.10, 20.40.70
   Hazards (see "disk data file, hazards")
   Hexadecimal numbers: 30.20.00, 45.05.30
   High/low/equal compare: 70.40.20
   IBM System/360: 45.50.00
   IBM systems area: 60.10.20, 60.20.20
   ICOMP: 70.10.30
   IFIX: 70.40.10
   IF statement: 30.20.00
   INTEGER: 65.10.40
   ILS01: 65.10.40
   ILS02: 65.10.40
   ILS03: 65.10.40
   ILS04: 65.10.40
   Index: 25.40.20
   Index to a disk data file: 85.10.01, 85.10.20
   Maintaining: 85.10.20
   Indexed sequential (see "disk data file, organization, indexed sequential")
   Input data, errors: 15.20.70
   Input data: 55.20.00
   Inquiry: 15.10.40
   Insertion: 75.30.10
   Inside controls: 20.10.20
   Installation: 65.01.00, 05.30.00
   Integer arithmetic: 70.10.10
   Integers: 70.10.10, 80.60.00
   One-word: 80.50.00, 80.60.00
   Interchangeable chain cartridge: 20.20.10
   Internal sort: 75.10.00
   Interrupt: 70.20.01
Inventory: 10.40.40, 85.10.20
Involvement: 70.20.20
*ICCS card: 65.10.30, 70.20.20, 70.20.20, 90.30.10
IOND: 70.20.10
JOB: 55.10.00
Job management: 50.01.00
Job-to-job transition: 50.01.00
KEYBD: 70.20.10, 70.20.20
Keyboard, console: 45.10.10
Keypunching (see "cards, punching")
Key-tag pair: 75.10.00
Key verification: 20.10.20
Language selection: 20.60.01
Large real numbers: 70.10.20
LET (see "Location Equivalence Table")
Light pen: 45.35.00
Linear Programming: 20.60.01
LINK: 65.10.60, 70.20.20, 90.20.20, 90.30.40
LINK area: 65.10.50
Load on call: 65.10.40
LOCAL: 65.10.20, 65.10.30, 70.50.20, 85.10.10, 90.20.20, 90.20.30, 90.30.40
LOCAL area: 65.10.40
Location Equivalence Table: 50.01.00, 60.10.40, 80.30.20
Magnitude: 70.10.20
Main line: 25.30.20
Manpower requirements: 40.30.00
Master cartridge: 50.01.00
Matching: 20.10.20
Match/no match: 70.40.20
Mechanism Design System: 20.60.01
Merge order: 75.10.00
Merging: 75.10.00, 75.30.10
Minsend: 70.10.30
Modular programs: 15.20.50, 25.30.20
Monitor control record: 55.10.00
MOVE: 25.40.30, 70.40.20
MPY: 70.10.30
NCOMP: 70.40.20
Next record number indicator: 80.30.10, 80.40.10
Non-systems cartridge: 50.01.00
NSIGN: 70.10.30
Numbers
Binary: 45.05.30
Hexadecimal: 30.20.00, 45.05.30
Numerical Surface Techniques: 20.60.01
NZONE: 70.40.20
Object code: 70.50.01
Operation manual: 15.20.70
Optical reader: 45.40.00
Optical System Design: 20.60.01
Order entry: 45.40.00
Outside controls: 20.10.20
Overlap: 70.20.01
Overlays: 65.10.30
PACK: 70.30.00, 70.40.10
Packing factor: 80.40.00
Paper tape punch: 45.25.00
Paper tape reader: 45.25.00
PAPTZ: 65.10.30
Parallel operations: 40.30.00
Partitioned direct (see "disk data file, organization, partitioned direct")
Pass: 75.10.00
Patches: 30.40.00
PAUSE: 30.20.00, 45.05.30, 65.10.30, 70.20.10
Payroll: 10.40.60, 15.10.20, 15.20.10, 15.20.50, 55.30.00, 80.60.00, 85.10.30, 85.30.10
PDUMP: 30.20.00
Performance (see "running time, factors affecting")
Personnel: 05.01.00
Petroleum Engineering and Exploration: 20.60.01
PID (see "Program Information Department")
Pigeonhole sorting: 75.30.10
Pilot operation: 40.30.00
Planning: 05.10.00
  For conversion: 40.10.00
  For testing: 30.01.00
Plotter: 45.30.00
PUNCH: 65.10.30
Preinstallation: 05.01.00
PRINT: 70.20.10, 70.20.20
Printer, console: 45.05.10
Printers: 45.15.00
Priority interrupt system: 70.20.01
PRNTZ: 65.10.30
PRNZ: 65.10.30
Processing, order: 15.10.10
Programmers, experience: 15.10.90
Program Area: 65.10.50
Change authorization: 25.30.20
Changes: 25.10.00, 25.30.20
Comments: 25.40.10
Modular: 15.20.50, 25.30.20
Patches: 30.40.00
Testing: 30.01.00, 30.10.00
Type I: 20.60.01
Type II: 20.60.01
Type III: 20.60.01
Type IV: 20.60.01
Program Information Department: 50.01.00
Program Information manual: 35.10.10
Programming
Aids: 25.30.10
Modular: 25.30.20
Standards: 25.10.00
Project Control System: 20.60.01
PUNCH: 70.20.10, 70.20.20
Punched card systems: 10.20.00
Punching cards (see "cards, punching")
PUT: 25.40.50, 70.10.20, 70.30.00, 70.40.10
Random file organization (see "disk data file, organization, random")
READ: 70.20.10, 70.20.20
Read/write heads: 45.10.00, 80.10.00
READZ: 65.10.30
Real arithmetic: 70.10.20
Real numbers: 80.60.00
Output of large: 70.10.20
Multiplication of large: 70.10.20
Record
Layout: 30.40.00
Length: 80.40.00, 80.40.10
Length, computing: 80.50.00
Length, shortening: 80.60.00
Number: 85.10.30
Size: 15.10.70
Records: 80.20.00
Recovery: 15.20.60
Replacement selection: 75.30.10
Resident monitor: 65.10.10
Rotational delay: 45.10.00
Rounding: 70.10.20
Route accounting: 20.60.01
Route slip: 20.10.20
Run book: 15.20.70
Running time, factors affecting: 80.10.00, 80.40.00, 85.10.20, 85.10.30
SAC (see "storage access channel")
Sales analysis: 10.40.30
Sample documents: 10.10.00
Satellite cartridge: 50.01.00
SCA (see "synchronous communications adapter")
Scientific Subroutine Package: 20.60.01
Scratch disk: 50.01.00
SDFIO: 65.10.30
SDFND: 65.10.30, 70.50.20
Searching an index: 85.10.10
Sectors: 45.10.00, 80.10.00, 80.40.00
Utilization: 80.40.00
Seek time: 45.10.00
Selective sorting: 75.30.10
Selective tracing: 30.20.00
Sequential organization (see “disk data file, organization, sequential”)
Serial numbering: 20.10.20
SFIO: 65.10.30
SKIP: 70.20.10, 70.20.20
SOCAL: 65.10.10, 65.10.20, 70.50.20, 85.10.10, 90.10.10, 90.20.20, 90.20.30, 90.30.40
SOCAL area: 65.10.30
Sorting: 15.10.10, 15.10.20, 75.01.00, 75.10.00, 85.30.10
Mechanical: 70.40.20, 75.20.20
1130 flowchart: 75.40.00
Sort phases: 75.10.00
SQRT: 70.50.20
Stabilization time: 45.10.00
STACK: 70.20.10
Stacker select: 25.40.30
Standard precision: 80.50.00, 80.60.00
Arithmetic: 70.10.20
Statistical System: 20.60.01
Stock status: 15.10.40
STOP: 45.05.30, 70.20.10
Storage access channel: 45.45.00
Storage costs: 15.10.80
*STORE: 60.30.20
*STORECI: 65.10.40
Store core image: 60.30.20
*STOREDATA: 80.30.20, 80.70.10
Store data core image: 60.30.20
Strategy, testing: 30.10.00
STRESS: 20.60.01
String: 75.10.00
SUB: 55.10.00
Subprograms, subtypes of: 65.10.10
Subroutines: 25.30.20, 70.30.01
Devices not on your system: 60.20.20
Logarithmic: 60.20.20
Long argument lessons: 70.50.10
Unlikely to be used: 60.20.20
Subtrahend: 70.10.30
SUFO: 65.10.30
Supervisor program: 50.01.00
Survey: 10.10.00
Survey questionnaire
Accounts payable: 10.40.50
Accounts receivable: 10.40.20
Billing: 10.40.10
Inventory: 10.40.40
Payroll: 10.40.40
Sales analysis: 10.40.30
Synchronous communications adapter: 45.50.00
System overlay: 65.10.30
Systems cartridge: 50.01.00
Systems testing: 30.10.00
Table lookup: 25.10.00
Tag: 75.10.00
Tag sort: 75.10.00, 75.30.20
Teleprocessing: (see “synchronous communications adapter”)
Temporary indicator: 55.10.00
Test decks: 30.10.00
Testing: 30.10.00, 30.20.00, 30.30.00, 30.40.00
Throughput (see “running time, factors affecting”)
Time
Rotational delay: 45.10.00
Stabilization: 45.10.00
Stamps: 20.10.20
Timing: 70.60.10
Trace: 30.20.00, 30.30.00, 45.05.20, 70.10.20, 70.20.20, 70.50.20
Transfer vector: 65.10.10
Transmitting slip: 20.10.20
TSTOP: 30.20.00
TSTRT: 30.20.00
Type Composition: 20.60.01
TYPER: 70.20.10, 70.20.20
TYPEZ: 65.10.30
UA (see “user area”)
UDISK: 65.10.30
UNPAC: 70.30.00, 70.40.10
Unused area: 65.10.70
User area: 60.10.40, 60.20.20, 80.30.20, 80.70.10
Variable precision arithmetic (see “arithmetic, variable precision”)
Variable summary sheet: 25.30.10
Verifier: 20.10.10
WHOLE: 25.40.40, 70.10.20
Work Measurement Aids: 20.60.01
Working storage: 60.10.30, 60.20.20, 80.30.20, 80.70.10
WRTYZ: 65.10.30
WS (see “working storage”)
X-punch: 70.40.20
Zone balance: 20.10.20, 25.40.40
Zone punch: 20.10.20, 70.40.10, 70.40.20
1DUMY: 60.20.10
11-punch: 70.20.10, 70.40.20
11-zone: 70.40.20
12-punch: 70.40.20
12-zone: 70.40.20
941 report: 25.40.70
1055 Paper Tape Punch: 45.25.00
1130 Configurator: 45.55.00
1131 Central Processing Unit: 90.30.20
1132 Printer: 45.05.10, 45.15.00, 70.20.01, 90.30.20
1403 Printer: 45.05.10, 45.15.00, 70.20.01, 90.30.20
1442 Card Read Punch: 45.20.00, 70.20.10, 90.30.20
Model 5 Card Punch: 45.20.00
1627 Plotter: 45.30.00, 90.30.20
2250 Display Unit: 45.35.00
2315 Disk Cartridge: 45.10.00, 80.10.00
2501 Card Reader: 45.20.00, 90.30.20
Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you wish a reply, be sure to include your name and address.

COMMENTS

• Thank you for your cooperation. No postage necessary if mailed in the U.S.A. FOLD ON TWO LINES, STAPLE AND MAIL.
YOUR COMMENTS PLEASE...

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

Please note that requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or the IBM branch office serving your locality.