This program of instruction and various instructional materials for a secondary-postsecondary level course for programmer/analysts is one of a number of military-developed curriculum packages selected for adaptation to vocational instruction and curriculum development in a civilian setting. The eight-week, three-section course is designed to provide the skill to program electronic computers structured in COBOL and to code job streams using IBM, DOS, JCL, and utility programs. The program of instruction suggests a time schedule and gives the learning objective and reference(s) for each topic. Section 1, Data Representation, consists of a programmed text covering the binary, octal, and lexadecimal systems. Section 2, Basic COBOL Programming, contains a text and problem exercises with some answers. Topic areas include processing and updating a sequential file, producing an edited report and a report with calculations, processing external and internal tables, and debugging syntax errors. Section 3, Operating Systems, includes a text and programmed text. It focuses on disk operating systems (DOS) organization and operation. Specific topics include coding DOS Job Streams, DOS Librarian Programs, and DOS Utilities and Sorts.
This military technical training course has been selected and adapted by The Center for Vocational Education for "Trial Implementation of a Model System to Provide Military Curriculum Materials for Use in Vocational and Technical Education," a project sponsored by the Bureau of Occupational and Adult Education, U.S. Department of Health, Education, and Welfare.
MILITARY CURRICULUM MATERIALS

The military-developed curriculum materials in this course package were selected by the National Center for Research in Vocational Education Military Curriculum Project for dissemination to the six regional Curriculum Coordination Centers and other instructional materials agencies. The purpose of disseminating these courses was to make curriculum materials developed by the military more accessible to vocational educators in the civilian setting.

The course materials were acquired, evaluated by project staff and practitioners in the field, and prepared for dissemination. Materials which were specific to the military were deleted, copyrighted materials were either omitted or approval for their use was obtained. These course packages contain curriculum resource materials which can be adapted to support vocational instruction and curriculum development.
The National Center Mission Statement

The National Center for Research in Vocational Education's mission is to increase the ability of diverse agencies, institutions, and organizations to solve educational problems relating to individual career planning, preparation, and progression. The National Center fulfills its mission by:

- Generating knowledge through research
- Developing educational programs and products
- Evaluating individual program needs and outcomes
- Installing educational programs and products
- Operating information systems and services
- Conducting leadership development and training programs

FOR FURTHER INFORMATION ABOUT Military Curriculum Materials
WRITE OR CALL
Program Information Office
The National Center for Research in Vocational Education
The Ohio State University
1960 Kenny Road, Columbus, Ohio 43210
Telephone: 614/486-3555 or Toll Free 800/848-4815 within the continental U.S.
(except Ohio)
**Military Curriculum Materials Dissemination Is . . .**

A practice to increase the accessibility of military-developed curriculum materials to vocational and technical educators.

This project, funded by the U.S. Office of Education, includes the identification and acquisition of curriculum materials in print form from the Coast Guard, Air Force, Army, Marine Corps, and Navy.

Access to military curriculum materials is provided through a "Joint Memorandum of Understanding" between the U.S. Office of Education and the Department of Defense.

The acquired materials are reviewed by staff and subject matter specialists, and courses deemed applicable to vocational and technical education are selected for dissemination.

The National Center for Research in Vocational Education is the U.S. Office of Education's designated representative to acquire the materials and conduct the project activities.

**Project Staff:**

- Wesley E. Budke, Ph.D., Director
- National Center Clearinghouse
- Shirley A. Chase, Ph.D.
- Project Director

---

**What Materials Are Available?**

One hundred twenty courses on microfiche (thirteen in paper form) and descriptions of each have been provided to the vocational Curriculum Coordination Centers and other instructional materials agencies for dissemination.

Course materials include programmed instruction, curriculum outlines, instructor guides, student workbooks, and technical manuals.

The 120 courses represent the following sixteen vocational subject areas:

<table>
<thead>
<tr>
<th>Subject Area</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agriculture</td>
<td>Food Service</td>
</tr>
<tr>
<td>Aviation</td>
<td>Health</td>
</tr>
<tr>
<td>Building &amp; Construction</td>
<td>Heating &amp; Air Conditioning</td>
</tr>
<tr>
<td>Trades</td>
<td>Machine Shop</td>
</tr>
<tr>
<td>Clerical Occupations</td>
<td>Management &amp; Supervision</td>
</tr>
<tr>
<td>Communications</td>
<td>Meteorology &amp; Navigation</td>
</tr>
<tr>
<td>Drafting</td>
<td>Photography</td>
</tr>
<tr>
<td>Electronics</td>
<td>Public Service</td>
</tr>
<tr>
<td>Engine Mechanics</td>
<td></td>
</tr>
</tbody>
</table>

The number of courses and the subject areas represented will expand as additional materials with application to vocational and technical education are identified and selected for dissemination.

**How Can These Materials Be Obtained?**

Contact the Curriculum Coordination Center in your region for information on obtaining materials (e.g., availability and cost). They will respond to your request directly or refer you to an instructional materials agency closer to you.

**CURRICULUM COORDINATION CENTERS**

**EAST CENTRAL**
- Rebecca S. Douglass
- Director
- 100 North First Street
- Springfield, IL 62777
- 217/782-0759

**MIDWEST**
- Robert Patton
- Director
- 1515 West Sixth Ave.
- Stillwater, OK 74704
- 405/377-2000

**NORTHEAST**
- Joseph F. Kelly, Ph.D.
- Director
- 225 West State Street
- Trenton, NJ 08625
- 609/292-6562

**NORTHWEST**
- William Daniels
- Director
- Building 17
- Airdustrial Park
- Olympia, WA 98504
- 206/753-0879

**SOUTHEAST**
- James F. Shill, Ph.D.
- Director
- Mississippi State University
- Drawer DX
- Mississippi State, MS 39762
- 601/325-2510

**WESTERN**
- Lawrence F. H. Zane, Ph.D.
- Director
- 1776 University Ave.
- Honolulu, HI 96822
- 808/948-7834
**PROGRAMMER/ANALYST COURSE**

532-74F1

Developed by:
United States Army

Occupational Area:
Clerical Occupations

Development and Review Dates:
September 1978

Target Audiences:
Grade 12 - Adult

Print Pages: 625
Microfiche: 11
Availability: Vocational Curriculum Coordination Centers

---

**Contents:**

<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>
</tr>
</thead>
<tbody>
<tr>
<td>Data Representation</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>Basic Cobol Programming</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>Operating Systems</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>

X Materials are recommended but not provided.
Course Description:

This course is designed to provide the skill to program electronic computers structured in Cobol and to code jobstreams using IBM, DOS, JCL, and utility programs. The course covers 8 weeks of instruction.

Data Representation consists of a programmed text covering the binary, octal and hexadecimal number systems.

Basic Cobol Programming section contains a text and problem exercises with some having answers. Topic areas include processing and updating a sequential file; producing an edited report and a report with calculations; processing external and internal tables; debugging syntax errors.

The Operating Systems segment focuses on disk operating system organization and operation. Specific topics of coding DOS Jobstreams, DOS Librarian Programs, DOS Utilities and Sorts are included.
<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program of Instruction</td>
<td>1</td>
</tr>
<tr>
<td>Data Representation Programmed Text</td>
<td>20</td>
</tr>
<tr>
<td>Basic Cobol Programming</td>
<td>148</td>
</tr>
<tr>
<td>Operating Systems</td>
<td>473</td>
</tr>
</tbody>
</table>
Annex F - Update a Sequential Master File Using a Sequential Transaction File

Terminal Learning Objective: Given program specifications, write a COBOL program to update a sequential master file with a transaction file containing adds, deletes, changes and errors. The program must also produce an update listing and a master file listing.

532-74F-1242 Establish Read Switches Type
Hours - ? 2C
Learning Objective: Using working storage, establish read switches to govern the access to each file.
Ref: Shelley/Cashman Structured COBOL

532-74F-1243 Update Master File Type
Hours - 29 7C, 22PE2
Learning Objective: Given program specifications, match records on two files to produce an updated output master.
Ref: Shelley/Cashman Structured COBOL

532-74F-1244 COBOL V Review and Exam Type
Hours - 8 4C, 4E
Learning Objective: N/A
Ref: N/A

Annex F Total Hours Allotted: 39
Annex E - Process External and Internal Tables in COBOL

Terminal Learning Objective: Given program specifications code COBOL routines to build tables from external or internal data and match input files to the completed table.

532-74F-1238 Establish Internal Table
Hours - 14 2C, 12PE2

Learning Objective: Using a month conversion table, write the COBOL statements to change the numeric month to the month's name.

Ref: Shelley/Cashman Structured COBOL

532-74F-1239 External Table Handling
Hours - 14 5.5C, 8.5PE2

Learning Objective: Given program specifications, code COBOL routines to build a table and search the table for match or no match condition.

Ref: Shelley/Cashman Structured COBOL

532-74F-1240 COBOL IV Exam and Review/Critique
Hours - 6 2C, 4E

Learning Objectives: N/A

Ref: N/A

Annex E Total Hours Allotted: 34

532-74F-1234 COBOL III Review and Exam
Hours - 8 4C, 4E

Learning Objective: N/A

Ref: N/A

Annex D Total Hours Allotted: 37
Annex D - Produce a Report with Calculations and Minor Control Breaks in COBOL

Terminal Learning Objective: Given program specifications, code a COBOL program that will perform arithmetic calculations and test for changes in input data to produce a report with group totals at a control break.

532-74F-1230 Test for Control Breaks
Hours - 12.5 Type

Learning objective: Given input control fields, code routines to process changes in the fields that will result in recognizing the change and processing each group of like items as individual entities.

Ref: Shelley/Cashman Structured COBOL

532-74F-1232 Advanced Condition Tests
Hours - 2 Type

Learning Objective: Given specifications, code routines that use condition names and nested IF statements syntactically and logically correct.

Ref: Shelley/Cashman Structured COBOL

532-74F-1232 Arithmetic Verbs
Hours - 6 Type

Learning Objective: Given specifications, code COBOL arithmetic verbs in conjunction with condition statements to produce minor totals at control breaks.

Ref: Shelley/Cashman Structured COBOL

532-74F-1233 Page Headers & Trailers
Hours - 8.5 Type

Learning Objective: Given report requirements, code COBOL routines that will result in multiple headings with the current data and sequential page numbers on output reports.

Ref: Shelley/Cashman Structured COBOL
Hours - 6

Learning Objective: N/A

Ref: N/A

Annex C Total Hours Allotted: 39
Annex C - Produce Edited Report with Minor Control Breaks in COBOL

Terminal Learning Objective: Given program specifications, code a COBOL program to read sequential input, perform condition tests, accumulate totals and use report editing to produce a report with a heading and total line.

532-74F-1220  Code Special Names Paragraph  Type

Hours -  .5  .5C

Learning Objective: Given the COBOL manual, code the special names paragraph correctly.

Ref: N/A

532-74F-1221  Establishing a Heading Line and Accumulations in the Data Division  Type

Hours -  1.5  1.5C

Learning Objective: Given program specifications code the Data Division entries to establish a heading line and accumulators.

Ref: Shelley/Cashman Structured COBOL

532-74F-1222  Code ADD and IF Verbs Logically Within the Procedure Division  Type

Hours -  14  4C, 10PE2

Learning Objective: Given program specifications, code the COBOL routines to print a heading, accumulate totals and to conditional tests.

Ref: Shelley/Cashman Structured COBOL

532-74F-1223  Editing Data and Formatting A Total Line  Type

Hours -  17  5C, 12PE2

Learning Objective: Given program specifications, write COBOL routines to edit numeric data and print a final total line.

Ref: Shelley/Cashman Structured COBOL
Learning Objective: Given program specifications, code a program to read a sequential file, move data and print a simple report.

Ref: Shelly/Cashman Structured COBOL

532-74F-1215 COBOL I Exam & Review/Critique
Hours - 7
Learning Objective: N/A
Ref: N/A

Annex B Total Hours Allotted: 37
Annex B - Process Sequential File in COBOL

<table>
<thead>
<tr>
<th>Code COBOL Identification Division</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours -</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>1C, .5PE2</td>
</tr>
</tbody>
</table>

Learning Objective: Given program specifications, code a COBOL Identification Division that is syntactically correct and that meets specifications.

Ref: Shelley/Cashman Structured COBOL

<table>
<thead>
<tr>
<th>Code COBOL Environment Division</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours -</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>1C, .5PE2</td>
</tr>
</tbody>
</table>

Learning Objective: Given program specifications, code a COBOL Environment Division that is syntactically correct and that meets specifications.

Ref: Shelley/Cashman Structured COBOL

<table>
<thead>
<tr>
<th>Code COBOL Data Division</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours -</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>4C, 4PE2</td>
</tr>
</tbody>
</table>

Learning Objective: Given program specifications, code a COBOL Data Division that is syntactically correct and that contains all entries necessary for correct program operation.

Ref: Shelly/Cashman Structured COBOL

<table>
<thead>
<tr>
<th>Code I/O and Movements Verbs</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours -</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>9C</td>
</tr>
</tbody>
</table>

Learning Objective: Given program specifications, code the COBOL verbs necessary to perform sequential I/O operations and to move data within memory.

Ref: Shelly/Cashman Structured COBOL

<table>
<thead>
<tr>
<th>Problem Solving Exam &amp; Review</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours -</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>3C, 4E</td>
</tr>
</tbody>
</table>

Learning Objective: N/A

Ref: N/A

Annex A Total Hours Allotted: 41
SECTION IV - ANNEXES

Annex A - Solve ADP Problems

Terminal Learning Objectives: Given a narrative problem, construct a systems flowchart, structured flowchart, HIPO and IPO that will represent a solution to the problem in structured terms.

531-74F-1201  Draw a Systems Flowchart  Type
Hours - 1.5 .5C, 1PE2

Learning Objective: Given the inputs and outputs to a program, draw a chart that will use the proper symbols to depict the flow of data in the system.

Ref: Class Notes

531-74F-1202  Draw a Hierarchy Chart  Type
Hours - 5 3C, 2PE2

Learning Objective: Given program specifications, draw a hierarchy chart that correctly reflects the logical structure of a program that will result in the required output.

Ref: Class Notes, Check Problem

532-74F-1203  Write an IPO Chart  Type
Hours - 15.5 6C, 9.5PE2

Learning Objective: Given a HIPO and program specifications correctly determine the inputs, outputs and processing required to meet the requirements.

Ref: Class Notes, PE 1 through PE 4

532-74F-1204  Draw a Structured Flowchart  Type
Hours - 12 4C, 8PE2

Learning Objective: Given program specifications, a HIPO and an IPO draw a structured flowchart that represents the logic flow within an IPO.

Ref: Class Notes, PE 1 through PE 4
A. Course: 532-74F1, Programmer/Analyst

B. Purpose: To provide enlisted personnel with the ability to program electronic computers in structured COBOL; to code job streams using IBM DOS JCL and utility programs and to work in the Army ADP environment interfacing with USACSC and other ADP agencies. Personnel are trained in MOS 74F10

C. Prerequisites:

1. Must have a course in high school algebra or score 45 or greater on GED test 5, high school level.
2. Standard score of 105 or better in aptitude area ST or CL.
3. Nine months or more of active duty remaining after course completion.
4. No security clearance required for this course.
5. AR 611-201, Special Requirements Section, requires a security clearance for MOS 74F.

D. Scope: Students learn ADP topics through in-class conference, practical exercises in COBOL, JCL and utilities and self-paced instruction.

E. Length: Peacetime 9 weeks, 3 days Mobilization 8 weeks, 3 days

F. Training Location: US Army Institute of Administration Fort Benjamin Harrison, Indiana 46216

G. MOS Feeder pattern:

<table>
<thead>
<tr>
<th>Prerequisite MOS</th>
<th>MOS trained in this course</th>
<th>Feeds Following MOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>09B</td>
<td>74F10</td>
<td>74F20, 74F30, 74F40, 74Z50</td>
</tr>
</tbody>
</table>

H. Ammunition Requirements: None.
Program of Instruction

For

Programmer/Analyst Course

532-74F1

Length: Peace time - 9 weeks, 3 days

Mobilization - 8 weeks, 3 days

Approved by:
Annex G - Data Representation

Terminal Learning Objective: Given the decimal, binary, and hexadecimal numbering systems, be able to describe the properties and manipulations of each and be able to manipulate the numbers within each number system.

Manipulate Number Systems

531-74F-1245

Hours - 2

Type IC, 1PE2

Learning Objective: Given several problems in the decimal, binary, and hexadecimal number systems, perform the indicated manipulations for each.

Ref: Data Representation Handout

Exam and Review/Critique

531-74F-1246

Hours - 2

Type 2E

Learning Objective: N/A

Ref: N/A

Annex G Total Hours Allotted: 4
Annex H - Debug COBOL Syntax Errors

Terminal Learning Objective: Given a COBOL listing with three different levels of syntax errors, determine all statements in error and correct them.

Exam and Review/Critique

Annex H Total Hours Allotted: 4
Annex I - Debug DOS Program Abends

Terminal Learning Objective: Given a COBOL compile listing with all options, a linkedit map and a core dump, find the error causing the program to abend and correct it.

531-74F-1250 Code a Simple ALC Program Type
Hours - 8 6C, 2PE2

Learning Objective: Given in-class notes, code and execute an ALC program to produce a calendar.

Ref: ALC handout

531-74F-1251 Correct DOS Abend Type
Hours - 10 3C, 7PE2

Learning Objective: Given compile and linkedit listings, and a core dump, find a statement in error and correct it.

Ref: PE handouts

531-74F-1252 Exam and Review/Critique Type
Hours - 4 2C, 2E

Learning Objectives: N/A

Ref: N/A

Annex I Total Hours Allotted: 22
### Annex J - Know DOS Organization and Operation

**Terminal Learning Objective:** Given the System Control and Service Manual, describe the function of each component of DOS as well as the benefits of operating systems and the effects of multiprogramming.

<table>
<thead>
<tr>
<th>Course Code</th>
<th>Title</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>532-74F-1255</td>
<td>Operating Systems</td>
<td></td>
</tr>
<tr>
<td>Hours</td>
<td>2</td>
<td>2C</td>
</tr>
</tbody>
</table>

**Learning Objective:** Given class reference materials, describe the purpose and advantages of an operating system naming at least 4 advantages.

Ref: OS Handout, System Control and Service

<table>
<thead>
<tr>
<th>Course Code</th>
<th>Title</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>532-74F-1256</td>
<td>Multiprogramming Concepts</td>
<td></td>
</tr>
<tr>
<td>Hours</td>
<td>2</td>
<td>2C</td>
</tr>
</tbody>
</table>

**Learning Objective:** With class notes, describe the effect multiprogramming has on total system throughput.

Ref: OS Handout

<table>
<thead>
<tr>
<th>Course Code</th>
<th>Title</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>532-74F-1257</td>
<td>DOS Libraries</td>
<td></td>
</tr>
<tr>
<td>Hours</td>
<td>2</td>
<td>2C</td>
</tr>
</tbody>
</table>

**Learning Objective:** Given class references and a list of programs, state which DOS library will store each program.

Ref: OS Handout  
System Control and Service  
DOS Job Control, Shelly/Cashman

Annex J Total Hours Allotted: 6
Annex K - Code DOS Jobstream

Terminal Learning Objective: Given all class reference material, program specifications and job requirements, code jobstreams to compile, link and/or execute programs in the DOS or DOS-E environment.

532-74F-1260 Code DOS Jobstream

Type

Hours - 16.5 9.5C, 7PE2

Learning Objective: Given all class reference material, program specifications and job requirements, code jobstreams to compile, link and/or execute programs in the DOS or DOS-E environment.

Ref: OS Handout
System Control & Service
DOS Job Control, Shelley/Cashman

532-74F-1261 Code DOS-E Extensions to DOS

Type

Hours - 3.5 2.5C, 1PE2

Learning Objective: Given a DOS Jobstream, revise the JCL to take advantage of all applicable DOS-E (ADAS, DYNAM-T) benefits.

Ref: Instruction Notes

532-74F-1262 Exam and Review/Critique

Type

Hours - 6 2C, 4E

Learning Objective: N/A

Ref: N/A

Annex K Total Hours Allotted: 26
Annex L - Code DOS Librarian Programs

Terminal Learning Objective: Given all class references, code the job-streams to display a library directory, display and/or punch a library entry and perform maintenance or any or all DOS libraries. The jobstream must be syntactically correct and must meet stated requirements.

532-74F-1265  DOS Library Service Routines  Type
Hours -  5.5  3C, 2.5PE2

Learning Objective: Given all class references, code a jobstream using SSERV, CSERV, RSERV and/or DSERV without syntax errors and with all stated requirements met.

Ref: OS Handout
    System Control & Service
    DOS Job Control, Shelley/Cashman

532-74F-1266  DOS Library Maintenance Routines  Type
Hours -  6.5  3.5C, 3PE2

Learning Objective: Given all class references, code a jobstream to perform any MAINT function on any library. The jobstream must be without syntax errors and must meet stated requirements.

Ref: System Control & Service
    DOS Job Control, Shelley/Cashman

Annex L Total Hours Allotted: 12
Annex M - Code DOS Utilities and Sorts

Terminal Learning Objective: Given all class references and job requirements, code a jobstream using file to file utilities and the SORT program. The jobstream must be syntactically correct and must produce output that meets stated requirements.

532-74F-1270  DOS File to File Utilities  Type
Hours - 6

Learning Objective: Given all class references and job requirements, code a jobstream to transfer data from one storage media to another. The jobstream must meet requirements and be syntactically correct.

Ref: DOS & TOS Utilities
OS Handout

532-74F-1270  DOS SORT/MERGE  Type
Hours - 9

Learning Objective: Given all class references and job requirements, code a jobstream to sort records into a given sequence. The jobstream must result in sorted output and must be syntactically correct.

Ref: DOS Tape and Dist SORT/MERGE Program

532-74F-1271  CSC Utilities  Type
Hours 1

Learning Objective: Given the CSC Utilities Manual index, find the desired utility within 3 minutes.

Ref: CSC Utilities Manual

532-74F-1272  Utilities Review & Exam  Type
Hours - 6

Learning Objective: N/A

Ref: N/A

Annex M Total Hours Allotted: 22
Annex N - Report Systems Problems to USACSC

532-74F-1275  Complete an Incident Report
Hours - 1.5

Learning Objective: Given a scenario and blank forms, complete an Incident Report (USACSC Form 53) with all required information.
Ref: DAP Interface With USACSC

532-74F-1276  Complete a Systems Change Request
Hours - 1.5

Learning Objective: Given a scenario and blank forms, complete a Systems Change Request with all required information.
Ref: DPA Interface With USACSC

532-74F-1277  CSC/DPA Interface Exam
Hours - 1

Learning Objective: N/A
Ref: N/A

Annex N Total Hours Allotted: 4
Annex 0 - Produce Systems Documentation

Terminal Learning Objective: Given sample DOD documentation standards and program specifications, compile an ADP documentation packet in the proper format and with all required information.

532-74F-1280 Programmer's Documentation

Hours - 6

Type 3C, 3PE2

Learning Objective: Given sample DOD standards and applicable specifications, compile program documentation in the right format and with all required information.

Ref: DOD Documentation Standards

532-74F-1281 Operations Documentation

Hours - 5

Type 3C, 2PE2

Learning Objective: Given sample DOD standards and applicable specifications, compile operations documentation in the right format and with all required information.

Ref: DOD Documentation Standards

532-74F-1282 Documentation Review and Exam

Hours - 4

Type 2C, 2E

Learning Objective: N/A

Ref: N/A

Annex 0 Total Hours Allotted: 15
DATA REPRESENTATION

PART I

BINARY NUMBER SYSTEM
READ THE FOLLOWING INSTRUCTIONS CAREFULLY BEFORE YOU START THE LESSON

MATERIALS REQUIRED:

In addition to the booklet you should have a pencil and some paper.

A quite environment that is conducive to strong concentration.

PROCEDURES THAT SHOULD BE FOLLOWED:

Read the frame very carefully.
In some cases the frame should be read twice.
Most of the frames require a response on the part of the student.
Write down the answer that you feel is correct before you turn
the page to the answer.
The answer will always be found at the top of the next page.
If your answer is correct continue to the next frame.
If your answer was not correct return to the frame and reread
inserting the correct answer as you read.

IF A FRAME MAKES NO SENSE AT ALL TO YOU RAISE YOUR HAND AND WAIT FOR
INSTRUCTOR.
INTRODUCTION

To converse with someone who speaks only a foreign language would require one of us to learn another language so we can communicate. Similarly, to communicate with a computer system it is necessary to learn and use its specific language.

OBJECTIVE

This Programmed Instruction Text covers the Binary Number System. Upon completion of the text, you will be able to:

1. Convert a Binary number to a Decimal number.
2. Convert a Decimal number to a Binary number.
3. Convert a Binary Coded Decimal to a Decimal number.
4. Convert a Decimal number to a Binary Coded Decimal.
5. Perform Binary arithmetic functions.
The number system taught in grade school is the Decimal Number System. In this number system ten symbols are used to represent digits.

What are the ten symbols used to indicate digits in the Decimal Number System?
ANS: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9

I hope you have not forgotten the Decimal Number System.

The BASE of a number system is the number of symbols used to indicate digits.

What is the BASE of the Decimal Number System?
ANS. Since ten symbols are used to represent the digits 0 through 9, the BASE of the Decimal Number System is TEN.

The positional values for three positional columns of the Decimal Number System are:

```
  100  10  1
```

The positional values of any number system are determined by taking the BASE of the number system and raising it to progressive powers substantially.

Progressive Powers

```
| 10^2 | 10^1 | 10^0 |
```

*Any number raised to the zero power is 1.*

Indicate the positional values for four columns of the Decimal Number System.
The Decimal Number 25 represented with the positional value of columns.

<table>
<thead>
<tr>
<th>1000</th>
<th>100</th>
<th>10</th>
<th>1</th>
</tr>
</thead>
</table>

or

| 10³ | 10² | 10¹ | 10⁰ |

Indicate the positional values over the Decimal Number 425.

| 10 | 1 |
| 2 | 5 |
By using a combination of positional values in the columns, a number can be represented. To determine the value of the number represented, the digit indicated in a column is multiplied by the positional value of the column; then, the values of the digits are added. For example:

The Decimal Number 425.

\[
\begin{array}{c|c|c|c}
100 & 10 & 1 \\
\hline
4 & 2 & 5 \\
\end{array}
\]

\[
\begin{align*}
5 \times 1 &= 5 \\
2 \times 10 &= 20 \\
4 \times 100 &= 400 \\
\end{align*}
\]

\[
425 \quad \text{Positional values of columns}
\]

Indicate how the value of Decimal Number 742 is determined.
The largest number that can be indicated in one column of any number system will be the base of the number system minus 1.

What is the largest number that can be indicated in one column of the Decimal Number System?
ANS.  9

BASE of Decimal Numbering System = 10
Minus 1
= -1

or

<table>
<thead>
<tr>
<th>1000</th>
<th>100</th>
<th>10</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>

What is the largest number that can be represented in two positional columns?
The electronic circuitry used in data processing systems function in binary states. This means that the circuitry can indicate only two possible states, either an "on" or "off" condition. This characteristic made it necessary to develop a number system that could be used by a computer. This number system is called the Binary Number System.

In this number system only the symbols 0 and 1 are used.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

What is the BASE of the Binary Number System?
The **BASE** of a number system is the **number** of symbols used to indicate digits. In the Binary Number System only two symbols, 0 and 1, are used.

If the **BASE** of the Binary System is **TWO**, the positional values for four columns are:

\[
\begin{array}{cccc}
2^3 & 2^2 & 2^1 & 2^0 \\
8  & 4  & 2  & 1 \\
\end{array}
\]

or

\[
\begin{array}{cccc}
2^3 & 2^2 & 2^1 & 2^0 \\
8  & 4  & 2  & 1 \\
\end{array}
\]

What are the positional values for **seven** columns of the Binary Number System?
What is the largest digit that can be indicated in one positional column in the Binary Number System?
ANS: 1

BASE of Binary Number System = 2

Minus 1 = -1

1

As in any number system, using a combination of positional values in the columns a number can be constructed or represented. A combination of 1's and 0's in this number system can represent a decimal number.
Method of converting Binary number to Decimal number.

Example is of Binary number 101 converted to a Decimal number.

1. Assign positional values of Binary Number System above the Binary number.

<table>
<thead>
<tr>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"1" = "on" condition  
"0" = "off" condition

2. Multiply digit indicated in each column by its positional value.

<table>
<thead>
<tr>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Positional values of columns

1 x 1 = 1
1 x 4 = 4

3. Add positional values of columns.

1 x 1 = 1
1 x 4 = 4
5

From this conversion example we have learned that the Binary number 101 is equivalent to the Decimal number 5. Dots under the Binary number indicate positional columns. The columns are represented internally in a computer with a bit that can be turned "on" or "off". So the dot may also represent a bit.

Convert the Binary number 1000111 to a Decimal number.

43
**ANS:** 71

<table>
<thead>
<tr>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
1 \times 1 &= 1 \\
1 \times 2 &= 2 \\
1 \times 4 &= 4 \\
1 \times 64 &= 64
\end{align*}
\]

71 (Decimal Number)

---

Convert Binary 1010101 to a Decimal Number.
Convert Binary 011001 to a Decimal Number.

1 x 1 = 1
1 x 4 = 4
1 x 16 = 16
1 x 64 = 64

85 (Decimal)

(Number)
ANS: 57

<table>
<thead>
<tr>
<th></th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

1 × 1 = 1
1 × 8 = 8
1 × 16 = 16
1 × 32 = 32

57 (Decimal)
(Number)

TEST

Convert following Binary numbers to Decimal. Each dot (.) under the Binary number indicates a positional column.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>DECIMAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. 101</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b. 10001</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c. 1001</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d. 111001</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Method to convert Decimal number to Binary number.

EXAMPLE: Decimal number 91 converted to a Binary number.

1. Assign positional values of the number system the Decimal number is to be converted to. The highest positional value must be larger than the Decimal number to be converted.

   Decimal number to be converted
   -----------------------------------------------
<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Indicate in the column the number of times the positional value will go into the Decimal number. "0" indicates positional value will not go into Decimal number. Begin with highest positional value.

   91 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
   ---|-----|----|----|----|---|---|---|---|
   | 0   |    |    |    |    |   |   |   |
   | 1   |    |    |    |    |   |   |   |
3. Multiply the positional value by the digit indicated in the column; subtract result from Decimal number.

4. Indicate the number of times the next positional value will go into the remainder.

The Decimal number 91 is equivalent to the Binary number 1011011.

Convert the Decimal number 14 to a Binary number.
Convert the Decimal number 72 to a Binary number.

Solution:

<table>
<thead>
<tr>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
14 & \triangleq (8 \times 1) \\
8 & \triangleq (8 \times 1) \\
6 & \\
4 & \triangleq (4 \times 1) \\
2 & \triangleq (2 \times 1) \\
2 & \\
0 & \\
\end{align*}
\]
Convert the Decimal number 45 to a Binary number.

ANS: 01001000
Convert following Decimal numbers to Binary.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. 4</td>
<td></td>
</tr>
<tr>
<td>b. 15</td>
<td></td>
</tr>
<tr>
<td>c. 26</td>
<td></td>
</tr>
<tr>
<td>d. 63</td>
<td></td>
</tr>
</tbody>
</table>
A Binary number like 010001101101100101 may be difficult to interpret because of the long string of 1's and 0's. To convert this Binary number to Decimal would involve assigning positional values to 20 positional columns, then adding their positional values. This would take a lot of time! For readability purposes and to save time, the Binary Coded Decimal was designed to easily convert a Binary number to a Decimal number. In Binary Coded Decimal (BCD) four bits are used to indicate one Decimal number. The positional values of these four bits are the same as the first four bits in the Binary Number System.

What are the positional values of the four bits in BCD?
The Decimal number 5 represented with four bits in Binary Coded Decimal (BCD).

```
<table>
<thead>
<tr>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

thus, 0 1 0 1
```

5

The Decimal number 45 represented in BCD:

```
<table>
<thead>
<tr>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

thus, 0 1 0 0 0 1 0 1

4 5

In BCD only the digits 0 through 9 can be indicated in one group of four bits.

Convert the Decimal number 79 to BCD.
Convert the Decimal number 458 to BCD.
Convert 3CD 0010 0110 to a Decimal number.
Convert BCD 0111 0011 0100 1001 to a Decimal number.
TEST

1. Convert Binary Coded Decimal (BCD) to Decimal.

<table>
<thead>
<tr>
<th>8421</th>
<th>8421</th>
<th>8421</th>
<th>100 10 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>0011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b.</td>
<td>....</td>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>c.</td>
<td>....</td>
<td>0101</td>
<td>0100</td>
</tr>
<tr>
<td>d.</td>
<td>0111</td>
<td>0110</td>
<td>1000</td>
</tr>
</tbody>
</table>

2. Convert Decimal to BCD.

<table>
<thead>
<tr>
<th>100 10 1</th>
<th>8421</th>
<th>8421</th>
<th>8421</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b.</td>
<td>1 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.</td>
<td>7 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d.</td>
<td>4 2 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
BINARY ADDITION

Rules applied when performing Binary addition:

a. $0 + 0 = 0$

b. $1 + 0$ or $0 + 1 = 1$

d. $1 + 1 = 0$ and carry 1 to next column to left

d. $1 + 1 + 1 = 1$ and carry 1 to next column to left

Addition problem: 

\[
\begin{array}{c}
010 \\
+ 011 \\
\hline
1 \\
\end{array}
\]

Add: 

\[
\begin{array}{c}
101 \\
+ 101 \\
\hline
101 \\
\end{array}
\]

\[
\begin{array}{c}
2 \\
+ 3 \\
\hline
5 \\
\end{array}
\]

\[
\begin{array}{c}
011 \\
\end{array}
\]

\[
\begin{array}{c}
1 \text{ carried} \\
\end{array}
\]
Solution:

\[
\begin{array}{c}
1 \ 1 \\
1 0 1 \\
1 0 1 \\
\hline
1 0 1 0
\end{array}
\]
Add:  
0110  
+ 1100  

ANS: 1000
Add:

0110101
+ 1110100

ANS: 10010
Perform Binary addition

a. 101  
   + 001  
   ---

b. 1000  
   + 1001  
   -----  

c. 1011  
   + 1001  
   -----  

d. 01110  
   + 1001  
   11011  

ANS: 10101001
The method by which subtraction is performed internally in a computer is **Subtraction by complementation**. The complement of a number is the amount that must be added to a certain number to have it indicate the largest number that can be indicated in one positional column. In Binary the complement of 1 is 0, the complement of 0 is 1 and, really, all we are doing is reversing the binary notation of the subtrahend (the number that is to be subtracted from another).
Subtraction problem: \[ 101 \]
\[ - 011 \]

a. Place the proper sign bit before each binary number; 0 = plus (+), 1 = minus (−). A plus sign is applied to minuend (top number) when minuend does not have plus or minus sign.

\[
\begin{array}{c|c|c}
101 & 0 & 101 \\
- 011 & 1 & 011 \\
\end{array}
\]

b. Complement subtrahend and add. Do not complement minuend or sign bits.

\[
\begin{array}{c|c|c}
0 & 101 & 0 & 101 \\
1 & 011 (\text{Subtrahend}) & 1 & 100 \\
1 & 0 & 001 \\
\end{array}
\]

Anything to left of sign bit is dropped or truncated. Thus 1 0 001 becomes 0 001.

c. Add 1 to result just obtained.

\[
\begin{array}{c|c|c}
0 & 001 & 0 & 001 \\
0 & 010 & \text{Sign bit is 0 or +. Thus answer is +2.}
\end{array}
\]

Subtract: \[ 100 \]
\[ - 010 \]
ANS: \[ \begin{array}{c|c|c}
0 & 010 \\
\hline
100 & 0 & 100 \\
- 010 & 1 & 101 \\
\hline
1 & 0 & 001
\end{array} \]

Solution: \[\begin{array}{c|c|c}
\hline
0 & 001 \\
\hline
+ 1
\hline
0 & 010 \text{ or } +2
\end{array} \]

Subtract: \[ \begin{array}{c|c|c}
1000 & \hline
- 0111 & \hline
\hline
-36
\end{array} \]
ANS: 0 0001

Subtract: 10110
- 01101

63

-37-
Subtract:

a. 0111
   - 0101
   = 0010

b. 1000
   - 0100
   = 0100

c. 1001
   - 0110
   = 0011

d. 10001
   - 01100
   = 00101
DATA REPRESENTATION

PART II

OCTAL NUMBER SYSTEM
READ THE FOLLOWING INSTRUCTIONS CAREFULLY BEFORE YOU START THE LESSON

MATERIALS REQUIRED:
In addition to the booklet you should have a pencil and some paper.
A quiet environment that is conducive to strong concentration.

PROCEDURE THAT SHOULD BE FOLLOWED:
Read the frame very carefully.
In some cases the frame should be read twice.
Most of the frames require a response on the part of the student.
Write down the answer that you feel is correct before you turn
the page to the answer.
The answer will always be found at the top of the next page.
If your answer is correct continue to the next frame.
If your answer was not correct return to the frame and reread
inserting the correct answer as you read.

IF A FRAME MAKES NO SENSE AT ALL TO YOU RAISE YOUR HAND AND WAIT FOR
INSTRUCTOR.
INTRODUCTION

To converse with someone who speaks only a foreign language would require one of us to learn another language so we can communicate. Similarly, to communicate with a computer system it is necessary to learn and use its specific language.

OBJECTIVE

This Programmed Instruction Text covers the Octal Number System. Upon completion of the text, you will be able to:

1. Convert an Octal number to a Decimal number.
2. Convert a Decimal number to an Octal number.
3. Convert Binary Coded Octal to a Decimal number.
4. Convert a Decimal number to Binary Coded Octal.
5. Perform Octal arithmetic functions.
OCTAL NUMBER SYSTEM

In the Octal Number System EIGHT symbols are used to represent its digits.

What is the BASE of the Octal Number System?
ANS: The BASE of the Octal Number System is EIGHT. This is because eight symbols are used to indicate digits in this number system.

Since the BASE of the Octal Number System is EIGHT, the positional values of the columns will be the BASE of the number system raised to progressive powers sequentially.

What is the positional value of the first column in the Octal Number System?
ANS: The first column will always be the BASE of the number system raised to the zero power. Any number to the zero power is one. Thus; First column = 8 or 1, which is indicated as 8^0 or 1

The positional values of three columns of the Octal number system are:

<table>
<thead>
<tr>
<th>8^2</th>
<th>8^1</th>
<th>8^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

What are the positional values for four columns of the Octal number system?
The largest number that can be indicated in one column of any number system will be the BASE of the number system minus 1.

What is the largest number that can be indicated in one column of the Octal Number System?
If the largest number that can be indicated in each column is 7, what are the symbols to indicate digits in the Octal Number System?
Method for converting OCTAL number system to DECIMAL number.

Example of the OCTAL number 225 converted to a DECIMAL number.

1. Assign positional values of OCTAL number system above the OCTAL number.

<table>
<thead>
<tr>
<th>512</th>
<th>64</th>
<th>8</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

2. Multiply digit indicated in each column by its positional value.

   Positional values of columns
   
   - \(5 \times 1 = 5\)
   - \(2 \times 8 = 16\)
   - \(2 \times 64 = 128\)
3. Add positional values of columns

\[
\begin{align*}
5 \times 1 &= 5 \\
2 \times 8 &= 16 \\
2 \times 64 &= 128 \\
\hline
&= 149
\end{align*}
\]

Thus, the OCTAL number 225 is equivalent to the DECIMAL number 149.

Convert the OCTAL number 136 to a DECIMAL number.
Convert the Octal number 217 to a Decimal number.

ANS: DECIMAL 94

Solution:

\[
\begin{array}{c|c|c|c}
64 & 8 & 1 \\
1 & 3 & 6 \\
\end{array}
\]

\[
\begin{align*}
6 \times 1 &= 6 \\
3 \times 8 &= 24 \\
1 \times 64 &= 64 \\
\end{align*}
\]

\[94\]
Convert OCTAL number to DECIMAL number

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>143</td>
<td>7</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

Solution:

\[
\begin{align*}
7 \times 1 &= 7 \\
1 \times 8 &= 8 \\
2 \times 64 &= \frac{128}{143}
\end{align*}
\]

TEST

Convert OCTAL number to DECIMAL number

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>1</td>
<td>2</td>
<td>. .</td>
</tr>
<tr>
<td>b.</td>
<td>1</td>
<td>7</td>
<td>. .</td>
</tr>
<tr>
<td>c.</td>
<td>3</td>
<td>2</td>
<td>. .</td>
</tr>
<tr>
<td>d.</td>
<td>7</td>
<td>6</td>
<td>. .</td>
</tr>
</tbody>
</table>
STEPS TO CONVERT DECIMAL NUMBER TO OCTAL NUMBER.

The Decimal number 89 converted to an Octal number.

1. Assign positional values of Octal Number System. The highest positional value must be larger than Decimal Number to be converted.

```
<table>
<thead>
<tr>
<th></th>
<th>512</th>
<th>64</th>
<th>8</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>89</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Decimal Number to be Converted

```
89
```

2. Indicate in the column the number of times the positional value will go into the Decimal number. "0" indicates positional value will not go into Decimal number. (Begin with highest positional value)

```
<table>
<thead>
<tr>
<th></th>
<th>512</th>
<th>64</th>
<th>8</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>89</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
3. If positional value in the column will go into the Decimal number, multiply the positional value that goes into the Decimal number by the digit indicated in the column; then, subtract result from the Decimal number.

\[
\begin{array}{c|c|c|c}
512 & 64 & 8 & 1 \\
0 & 1 & & \\
\end{array}
\]

\[
89 \quad 64 \quad 25
\]

\[
1 \times 64
\]

Thus, the Decimal number 89 is equivalent to the Octal number 131.

4. Indicate in the following columns the number of times the positional values will go into remainder until remainder is zero.

\[
\begin{array}{c|c|c|c}
512 & 64 & 8 & 1 \\
0 & 1 & 3 & 1 \\
\end{array}
\]

\[
89 \quad 64 \quad 25
\]

\[
1 \times 64
\]

\[
3 \times 8
\]

\[
1 \times 1
\]

Thus, the Decimal number 89 is equivalent to the Octal number 131.

Convert the Decimal number 27 to an Octal number.
Convert the Decimal number 42 to an Octal number.
Convert the Decimal number 205 to an Octal number.
Convert Decimal to Octal

<table>
<thead>
<tr>
<th>Dec</th>
<th>Oct</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dec</th>
<th>Oct</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Solution:

- **ANS:** OCTAL 315
- **512**
- **64**
- **8**
- **1**
- **0**
- **3**
- **1**
- **5**

- **205**
- **192**
- **13**
- **8**
- **5**
- **0**

- **(3 x 64)**
- **(1 x 8)**
- **(5 x 1)**

**TEST**

```
Convert Decimal to Octal

<table>
<thead>
<tr>
<th>10</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>6</td>
</tr>
<tr>
<td>b.</td>
<td>1 8</td>
</tr>
<tr>
<td>c.</td>
<td>4 6</td>
</tr>
<tr>
<td>d.</td>
<td>6 3</td>
</tr>
</tbody>
</table>

```
BINARY CODED OCTAL (BCO)

Binary Coded Decimal one decimal number was represented with 1's and 0's in a group of four bits (four columns). In Binary Coded Octal, three bits are used to represent one Octal number with 1's and 0's. The positioned values of these three bits are the same as the first three bits in the Binary Numbering System.

What are the positional values of the three bits in Binary Coded Octal?
The OCTAL number 6 is represented with three bits in Binary Coded Octal:

\[
\begin{array}{c|c|c|c}
4 & 2 & 1 \\
\hline
1 & 1 & 0 \\
\end{array}
\]

thus, \[110\] 6

The OCTAL number 36 represented in Binary Coded Octal.

\[
\begin{array}{c|c|c|c}
4 & 2 & 1 \\
\hline
0 & 1 & 1 \\
\end{array}
\quad \begin{array}{c|c|c|c}
4 & 2 & 1 \\
\hline
1 & 1 & 0 \\
\end{array}
\]

thus, \[011110\] 3 6

In Binary Coded Octal only the digits 0 through 7 can be indicated in one group of three bits.

Convert the Binary Coded Octal number 010100 to an Octal number.
Convert the Binary Coded Octal number 1 0 1 0 0 1 to an Octal number.

ANS: 24

Solution:

\[
\begin{array}{c}
0 & 1 & 0 \\
\hline
2 \\
\end{array}
\quad \begin{array}{c}
1 & 0 & 0 \\
\hline
4 \\
\end{array}
\]
Convert OCTAL number 43 to Binary Coded Octal.
Convert OCTAL number 72 to Binary Coded Octal.
TEST

1. Convert Binary Coded Octal to Octal number.

<table>
<thead>
<tr>
<th>4 2 1</th>
<th>4 2 1</th>
<th>8 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. 0 0 1</td>
<td>0 0 1</td>
<td>. . (Octal number)</td>
</tr>
<tr>
<td>b. 0 0 1</td>
<td>1 0 1</td>
<td>. . (Octal number)</td>
</tr>
<tr>
<td>c. 1 0 1</td>
<td>1 1 0</td>
<td>. . (Octal number)</td>
</tr>
<tr>
<td>d. 1 1 1</td>
<td>1 0 0</td>
<td>. . (Octal number)</td>
</tr>
</tbody>
</table>

2. Convert Octal number to Binary Coded Octal.

<table>
<thead>
<tr>
<th>6 4 8 1</th>
<th>4 2 1</th>
<th>4 2 1</th>
<th>4 2 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. 5</td>
<td>. .</td>
<td>. .</td>
<td>. .</td>
</tr>
<tr>
<td>b. 2 3</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>c. 5 2</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>d. 1 2 7</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
</tbody>
</table>
OCTAL ADDITION:

\[
\begin{array}{c}
43 \\
+ 55 \\
\hline
98
\end{array}
\]

1. Add one column at a time.

\[
\begin{array}{cc}
4 & 3 \\
+ & 5 & 6 \\
\hline
& 9 & 1
\end{array}
\]

2. If result is 7 or less indicate result in column. But if result is more than 7, subtract result by 8 (the base of the Octal number system).

\[
\begin{array}{c}
4 & 3 \\
+ & 5 & 6 \\
\hline
3 & +6 \\
\hline
9 \\
- 8 \\
\hline
1
\end{array}
\]
3. Indicate remainder in the column added and carry 1 to next column.

First column:

\[
\begin{array}{c}
1 \text{ carry} \\
\hline
1 & 4 & 3 \\
5 & 6 \\
\hline
1
\end{array}
\]

4. Add following columns in same manner (include 1 carry).

Second column:

\[
\begin{array}{c}
1 \text{ carry} \\
\hline
1 & 4 & 3 \\
5 & 6 \\
\hline
2 & 1 \\
\hline
\end{array}
\]

Third column:

\[
\begin{array}{c}
1 & 1 \\
4 & 3 \\
5 & 6 \\
\hline
1 & 2 & 1
\end{array}
\]

Thus, the result of the OCTAL addition is 121.

Perform OCTAL addition:

\[
\begin{array}{c}
2 \\
+ \ 7 \\
\hline
9
\end{array}
\]
Perform OCTAL addition:  
\[ 42 + 55 \]

\[ 63 \]
Perform OCTAL addition:

2
a. \( + 5 \)

b. \( + 7 \)

52
c. \( + 66 \)

64
d. \( + 31 \)
OCTAL SUBTRACTION

\[
\begin{array}{c}
45 \\
- 17
\end{array}
\]

1. Subtract one column at a time.

\[
\begin{array}{c}
4 \quad 5 \\
- 1 \quad 7 \\
\hline
\quad 1 \quad 8
\end{array}
\]

2. If minuend (number to be subtracted from) is smaller than subtrahend, borrow the base (8) from next column.

\[
\begin{array}{c}
3 \quad 1 \quad 3 \\
4 \quad 5 \\
- 1 \quad 7 \\
\hline
\quad 6 \quad 7
\end{array}
\]

\((5 + 8 \text{ (Base)})\)
3. Indicate result in the column subtracted

   **First Column:**
   \[
   \begin{array}{ccc}
   3 & 5 & 13 \\
   \hline
   17 & 7 \\
   \hline
   6 & 6 \\
   \end{array}
   \]

4. Subtract following column in same manner

   **Second Column:**
   \[
   \begin{array}{ccc}
   3 & 5 & 3 \\
   \hline
   17 & 1 \\
   \hline
   26 & 2 \\
   \end{array}
   \]

   Thus, the result of the OCTAL subtraction is 26.
Perform OCTAL subtraction:

\[
\begin{array}{c}
73 \\
-35 \\
\hline
38
\end{array}
\]

ANS: 17
Perform OCTAL subtraction:

5 2
a. -1 4

3 7
c. -1 7

6 4
b. -2 5

6 6
d. -5 7
DATA REPRESENTATION

PART III

HEXADECIMAL NUMBER SYSTEM
READ THE FOLLOWING INSTRUCTIONS CAREFULLY BEFORE YOU START THE LESSON

MATERIALS REQUIRED:

In addition to the booklet you should have a pencil and some paper.

A quiet environment that is conducive to strong concentration.

PROCEDURE THAT SHOULD BE FOLLOWED:

Read the frame very carefully.
In some cases the frame should be read twice.
Most of the frames require a response on the part of the student.
Write down the answer that you feel is correct before you turn the page to the answer.
The answer will always be found at the top of the next page.
If your answer is correct continue to the next frame.
If your answer was not correct return to the frame and reread
inserting the correct answer as you read.

IF A FRAME MAKES NO SENSE AT ALL TO YOU RAISE YOUR HAND AND WAIT FOR INSTRUCTOR.
To converse with someone who speaks only a foreign language would require one of us to learn another language so we can communicate. Similarly, to communicate with a computer system it is necessary to learn and use its specific language.

OBJECTIVE

This Programmed Instruction Text covers the Hexadecimal Number System. Upon completion of the text, you will be able to:

1. Convert a Hexadecimal number to a Decimal number.
2. Convert a Decimal number to a Hexadecimal number.
3. Convert Binary Coded Hexadecimal to a Decimal number.
4. Convert a Decimal number to Binary Coded Hexadecimal.
5. Perform Hexadecimal arithmetic functions.
HEXADECIMAL NUMBER SYSTEM

In the Hexadecimal Number System SIXTEEN symbols are used to represent its digits.

What is the BASE of the Hexadecimal Number System?
ANS: The BASE of the HEXADECIMAL number system is SIXTEEN. This is because sixteen symbols are used to represent digits in this number system.

Since the BASE of the HEXADECIMAL number system is SIXTEEN, the positional values of the columns will be the BASE of the number system raised to progressive powers sequentially.

What is the positional value of the first column in HEXADECIMAL number system?
The first column will always be the BASE of the number system raised to the zero power. Any number to the zero power is one. Thus first column = -16° or 1.

The positional values of three columns of the HEXADECIMAL number system are:

| 16^2 | 16^1 | 16^0 | or | 256 | 16 | 1 |

What are the positional values for four columns of the HEXADECIMAL number system?
The largest number that can be indicated in one column of any numbering system will be the base of the numbering system minus 1.

What is the largest number that can be indicated in one column of the hexadecimal number system?
ANS: 15  Solution: BASE of Hexadecimal Number System = 16
Minus 1 = \(-\frac{1}{15}\)

In the HEXADECIMAL number system alphabetic letters are used to indicate 10 through 15.

10 = A
11 = B
12 = C
13 = D
14 = E
15 = F

If the largest digit that can be indicated in each column is F (which is 15), what are the symbols that are used to indicate digits in the Hexadecimal number system?
Method for converting HEXADECIMAL number system to DECIMAL number system.

Example of the HEXADECIMAL number 4F converted to a DECIMAL number.

1. Assign positional values of HEXADECIMAL number system above the HEXADECIMAL number.

\[
\begin{array}{c|c|c}
256 & 16 & 1 \\
2 & 4 & F \\
\end{array}
\]

2. Multiply digit indicated in each column by its positional value.

\[
\begin{align*}
256 & \quad 16 & \quad 1 \\
\downarrow & \quad \downarrow & \quad \downarrow \\
15 & \times & 1 = 15 \\
4 & \times & 16 = 64
\end{align*}
\]
3. Add positional values of columns

\[
\begin{align*}
15 \times 1 & = 15 \\
4 \times 16 & = 64 \\
\hline
\end{align*}
\]

Thus, the \textsc{HexaDeciMAl} number \textit{4F} is equivalent to the \textsc{DeciMAl} number \textit{79}.

Convert \textsc{HexaDeciMAl} \textit{1B} to a \textsc{DeciMAl} number.
Convert HEXADECIMAL A3 to a DECIMAL number?

\[ 11 \times 1 = 11 \]
\[ 1 \times 16 = \frac{16}{27} \]

ANS: 27

Solution:

\[
\begin{array}{c|c|c|c}
256 & 16 & 1 & B \\
\hline
1 & 1 & & \\
\hline
\end{array}
\]

\[ 11 \times 1 = 11 \]
\[ 1 \times 16 = \frac{16}{27} \]
Convert HEXADECIMAL to DECIMAL.

<table>
<thead>
<tr>
<th>256</th>
<th>16</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td></td>
<td>E</td>
</tr>
<tr>
<td>b.</td>
<td>A</td>
<td>9</td>
</tr>
<tr>
<td>c.</td>
<td>F</td>
<td>3</td>
</tr>
<tr>
<td>d.</td>
<td>1</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>100</th>
<th>10</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Method for converting DECIMAL number system to HEXADECIMAL number system.

Example of the DECIMAL number 167 converted to a HEXADECIMAL number.

1. Assign positional values of HEXADECIMAL number system. The highest positional value must be larger than DECIMAL number to be converted.

   | 256 | 16 | 1 |
---|-----|----|---|
Decimal number to be converted 167

2. Indicate in the column the number of times the positional value will go into the DECIMAL number. "0" indicates positional value will not go into DECIMAL number (Begin with highest positional value.)

   | 256 | 16 | 1 |
---|-----|----|---|
167

   | 0    |    |   |
---|------|----|---|

3. If positional value in the column will go into the DECIMAL number, multiply the positional value that goes into the DECIMAL number by the digit indicated in the column; then, subtract result from the DECIMAL number.

\[
\begin{array}{c|ccc}
\text{256} & \text{16} & \text{1} \\
\hline
\text{0} & \text{A} & \text{7}
\end{array}
\]

\[
\frac{167}{160} \quad 10 \times 16
\]

\[
\frac{160}{7}
\]

4. Indicate in the following columns the number of times the positional values will go into remainder until remainder is zero.

\[
\begin{array}{c|ccc}
\text{256} & \text{16} & \text{1} \\
\hline
\text{0} & \text{A} & \text{7}
\end{array}
\]

\[
\frac{167}{160} \quad 10 \times 16
\]

\[
\frac{160}{7}
\]

\[
\frac{7}{7}
\]

\[
\frac{7}{0}
\]

Thus the DECIMAL number 167 is equivalent to the HEXADECIMAL number A7.

Convert the DECIMAL number 230 to a HEXADECIMAL number.
Convert the DECIMAL number 563 to a HEXADECIMAL number.
ANS: 233  
Solution:

\[
\begin{array}{c}
\text{563} \\
\text{512} \\
\text{51} \\
\text{48} \\
\text{3} \\
\text{3} \\
\text{0}
\end{array}
\]

\[
\begin{array}{c}
2 \times 256 \\
3 \times 16 \\
3 \times 1
\end{array}
\]

---

TEST

Convert Decimal to Hexadecimal

<table>
<thead>
<tr>
<th>a.</th>
<th>b.</th>
<th>c.</th>
<th>d.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>100</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>a.</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>b.</td>
<td>1</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>c.</td>
<td>9</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>d.</td>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>256</th>
<th>16</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

116
In Binary Coded Decimal, one decimal number was represented with 1's and 0's in a group of four bits (four columns). In Binary Coded Hexadecimal, again four bits are used to represent one Hexadecimal number with 1's and 0's. The positional values of these four bits are the same as the first four bits in the Binary Numbering System.

What are the positional values of the four bits in the Binary Coded Hexadecimal?
The Hexadecimal number A (10) represented in Binary Coded Hexadecimal (with four bits):

\[
\begin{array}{c|c|c|c|c|}
8 & 4 & 2 & 1 \\
\hline
1 & 0 & 1 & 0 \\
\end{array}
\]

thus, \(1010\).

A

The Hexadecimal number 6A represented in Binary Coded Hexadecimal.

\[
\begin{array}{c|c|c|c|c|}
8 & 4 & 2 & 1 \\
\hline
0 & 1 & 1 & 0 \\
\hline
1 & 0 & 1 & 0 \\
\end{array}
\]

thus, \(01101010\).

6 A

In Binary Coded Hexadecimal the digits 0 through F (15) can be represented in one group of four bits.

Convert the Binary Coded Hexadecimal number \(10110001\) to a Hexadecimal number.
Convert the Binary Coded Hexadecimal number 0 0 1 1 1 1 0 0 to a Hexadecimal number.
ANS: 3C  Solution: \[
\begin{array}{c}
0 & 0 & 1 & 1 \\
\hline
1 & 1 & 0 & 0 \\
\end{array}
\]

3  C

Convert Hexadecimal number 3D to Binary Coded Hexadecimal.
Convert Hexadecimal number BE to Binary Coded Hexadecimal.
ANS: 1011 1110

Solution:

B

E

1011

1110

TEST

1. Convert Binary Coded Hexadecimal to Hexadecimal.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>256 16 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>0111</td>
<td></td>
<td>. .</td>
</tr>
<tr>
<td>b.</td>
<td>0110</td>
<td>1101</td>
<td>. .</td>
</tr>
<tr>
<td>c.</td>
<td>1100</td>
<td>0110</td>
<td>. .</td>
</tr>
<tr>
<td>d.</td>
<td>0010</td>
<td>1011</td>
<td>1111</td>
</tr>
</tbody>
</table>

2. Convert Hexadecimal to Binary Coded Hexadecimal.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>C</td>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td>b.</td>
<td>E</td>
<td>4</td>
<td>. . .</td>
</tr>
<tr>
<td>c.</td>
<td>9</td>
<td>A</td>
<td>. . .</td>
</tr>
<tr>
<td>d.</td>
<td>2</td>
<td>B</td>
<td>. . .</td>
</tr>
</tbody>
</table>
HEXADECIMAL ADDITION:

A4
6E

1. Add one column at a time.

First Column:

A 4
6 E

4
+ 14
18

2. If result is F (15) or less indicate result in column (in Hexadecimal). But if result is more than F, subtract result by 16 (the base of the Hexadecimal number system).

First Column:

A 4
6 E

+ 14
18

- 16
 2

123
3. Indicate remainder in the column added and carry 1 to next column:

First Column:

\[
\begin{array}{c}
1 \\
\text{A} \\
\text{6} \\
\hline
\text{2}
\end{array}
\]

\[
\begin{array}{c}
4 \\
\hline
\text{14} \\
\hline
\text{18}
\end{array}
\]

\[
\begin{array}{c}
2 \\
\hline
\text{16} \\
\hline
\text{16} \\
\hline
\text{2}
\end{array}
\]

4. Add following columns in same manner (include 1 carry).

Second Column:

\[
\begin{array}{c}
1 \\
\text{A} \\
\text{6} \\
\hline
\text{2}
\end{array}
\]

\[
\begin{array}{c}
4 \\
\hline
10 \\
\hline
17
\end{array}
\]

\[
\begin{array}{c}
-16 \\
\hline
-16 \\
\hline
-1
\end{array}
\]

Third Column:

\[
\begin{array}{c}
1 \\
\text{A} \\
\text{6} \\
\hline
1
\end{array}
\]

Thus, the result of the Hexadecimal addition is HEXADECIMAL 112.
Perform Hexadecimal addition:

\[
\begin{array}{c}
B \\
+ 7 \\
\hline
13 \\
\hline
125
\end{array}
\]
Perform Hexadecimal addition:

\[
\begin{array}{c}
C \quad D \\
+ \quad B \quad E \\
\hline
-9 \quad 5 \quad 1 \quad 2 \quad 6
\end{array}
\]
Perform Hexadecimal addition:

a. $8 + 5$

b. $A6 + 45$

c. $B4 + 6E$

d. $DE + CF$
HEXADECIMAL SUBTRACTION

A5
+ 6D

1. Subtract one column at a time.

\[
\begin{array}{c}
\text{A}5 \\
\text{6}D \\
\hline
\text{5} \\
\hline
\text{-13}
\end{array}
\]

2. If minuend (number to be subtracted from) is smaller than subtrahend, borrow the base (16) from next column.

\[
\begin{array}{c}
9 \\
\text{A}5 \\
\text{6}D \\
\hline
\text{21} \\
\hline
\text{[5 + 16 (BASE)]}
\end{array}
\]
3. Indicate result in the column subtracted.

First Column:

\[
\begin{array}{c}
9 \\
A & 5 \\
6 & D \\
\hline
\end{array}
\begin{array}{c}
2 & 1 \\
- & 1 & 3 \\
\hline
8 & 8 \\
\end{array}
\]

4. Subtract following column in same manner.

Second Column:

\[
\begin{array}{c}
9 \\
A & 5 \\
6 & D \\
\hline
\end{array}
\begin{array}{c}
9 \\
- & 6 \\
\hline
3 & 3 \\
\end{array}
\]

Thus, the result of Hexadecimal subtraction is 38.
Perform Hexadecimal subtraction

\[
\begin{array}{c}
- \text{C3} \\
- \text{6B} \\
\text{18}
\end{array}
\]
Perform Hexadecimal subtraction:

a. \(41 - 2B\)  
b. \(A2 - 87\)  
c. \(C4 - A1\)  
d. \(DC - AC\)
DATA REPRESENTATION

PART IV

BCDIC and EBCDIC *

* An understanding of the Hollerith Code is a prerequisite for this part of Programmed Instruction Text. P.I. TEXT on Punched Cards is TC 14-71-50PT.
READ THE FOLLOWING INSTRUCTIONS CAREFULLY BEFORE YOU START THE LESSON

MATERIALS REQUIRED:

In addition to the booklet you should have a pencil and some paper.

A quiet environment that is conducive to strong concentration.

PROCEDURE THAT SHOULD BE FOLLOWED:

Read the frame very carefully.
In some cases the frame should be read twice.
Most of the frames require a response on the part of the student.
Write down the answer that you feel is correct before you turn
the page to the answer.
The answer will always be found at the top of the next page.
If your answer is correct continue to the next frame.
If your answer was not correct return to the frame and reread
inserting the correct answer as you read.

IF A FRAME MAKES NO SENSE AT ALL TO YOU RAISE YOUR HAND AND WAIT FOR
INSTRUCTOR.
INTRODUCTION

To converse with someone who speaks only a foreign language would require one of us to learn another language so we can communicate. Similarly, to communicate with a computer system it is necessary to learn and use its specific language.

OBJECTIVE

This Programmed Instruction Text covers the various codes used to represent alphabetic, special, and numeric data internally within a computer system. Upon completion of the text you will be able to:

1. Interpret the Binary Coded Decimal Interchange Code.
2. Interpret the Extended Binary Coded Decimal Interchange Code.
In Binary Coded Decimal, four bits were used to represent one decimal digit and only digits could be represented in this code. By extending this four bit field by two more bits we can represent alphabetic characters as well as numbers. This second generation six bit field is known as the BINARY CODED DECIMAL INTERCHANGE CODE (BCDIC).

Name the two punches required on a punched card to represent characters.
The zone and digit bits of the Binary Coded Decimal Interchange Code:

<table>
<thead>
<tr>
<th>Zone Bits</th>
<th>Digit Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA</td>
<td>8421</td>
</tr>
</tbody>
</table>

Zone bits used to represent zone punches.

<table>
<thead>
<tr>
<th>(Punch) Zone</th>
<th>(Bits) BA</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>01</td>
</tr>
</tbody>
</table>

What letters can be represented in the Hollerith Code with:

a. 12 zone punch?
b. 11 zone punch?
c. 0 zone punch?
What are the Hollerith Code punches used to represent the letter "B"?
ANS: "12" zone punch, "2" digit punch.

Indicate the letter "B" in the Binary Coded Decimal Interchange Code.
Indicate the letter "M" in BCDIC.
Indicate the letter "V" in BCDIC.
<table>
<thead>
<tr>
<th>B</th>
<th>A</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Indicate the number "6" in BCDIC.
ANS: \[
\begin{array}{c}
\text{BA8421} \\
\hline
\text{000110}
\end{array}
\]

**TEST**

Indicate character in BCDIC.

<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>BA8421</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. U</td>
<td>.......</td>
</tr>
<tr>
<td>b. D</td>
<td>.......</td>
</tr>
<tr>
<td>c. L</td>
<td>.......</td>
</tr>
<tr>
<td>d. 4</td>
<td>.......</td>
</tr>
</tbody>
</table>
Special characters used BCDIC will not be discussed in the Programmed Instruction Text. IBM System 360 Reference Data Card provides the bit configuration of special characters in BCDIC.
Whenever data is transferred from one device to another in a computer system, there is always a chance that a bit condition ("on" or "off") may be lost or gained. This of course, would change the character representation. An additional check bit position has been added to the six bit field (BCDIC) to make what is called a parity check so we can determine when a bit has been lost or gained:

```
check bit
C B A 8 4 2 1
```

When making an odd parity check, the number of "1" bits under the BAB421 are counted. If the number of "1" bits is even, another "1" bit is placed under the check bit (C) to make the entire number of 7 bits odd. For even parity the entire number of 7 bits is even.

The letter "C" with

1. Odd Parity:
   
   ```
   C B A 8 4 2 1
   1 1 1 0 0 1 1
   ```

2. Even Parity:
   
   ```
   C B A 8 4 2 1
   0 1 1 0 0 1 1
   ```

Place the proper check bit condition ("1" or "0") for the letter "B".

Check for odd parity:

```
C B A 8 4 2 1
  1 1 0 0 1 0
  . . . . . .
```
Place the proper check bit condition for the letter "E". Check for odd parity.

\[
\begin{array}{cccccccc}
C & B & A & 8 & 4 & 2 & 1 \\
0110010 \\
\end{array}
\]
Place the proper check bit condition for the letter "A". Check for even parity.

ANS: C B A 8 4 2 1
    1 1 1 0 1 0 1
    . . . . . .
Place the proper check bit condition for the letter "K". Check for even parity.

\[
\begin{array}{c}
C \ B \ A \ 8 \ 4 \ 2 \ 1 \\
1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \\
\end{array}
\]
Place proper check bit condition:

1. **Odd Parity**
   
   C B A 8 4 2 1
   
   a. - 1 1 0 1 0 0
   
   b. - 1 0 0 1 1 1
   
   c. - 1 1 1 0 0 1
   
   d. - 0 1 0 0 1 1

2. **Even Parity**
   
   C B A 8 4 2 1
   
   a. - 1 0 0 1 0 1
   
   b. - 0 1 0 1 1 1
   
   c. - 1 1 1 0 0 0
   
   d. - 1 0 0 1 0 0
In the Binary Coded Decimal Interchange Code, alphabetic characters could be represented as either capital letters (upper case) or small letters (lower case), but not in both cases. By extending the bit, we can represent both upper and lower cases of letters. The new third generation eight bit code is known as the Extended Binary Coded Decimal Interchange Code (EBCDIC).

<table>
<thead>
<tr>
<th>CASE BITS</th>
<th>ZONE BITS</th>
<th>DIGIT BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>21</td>
<td>8421</td>
</tr>
</tbody>
</table>
Case bits 1 1 indicate capital letters (upper case).
Case bits 1 0 indicate small letters (lower case).
Hexadecimal numbers are indicated by using 1's for both case & zone bits.

Representation of Characters:

<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>ZONE PUNCH</th>
<th>DIGIT PUNCH</th>
<th>CASE BITS</th>
<th>ZONE BITS</th>
<th>DIGIT BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>12</td>
<td>1</td>
<td>11</td>
<td>00</td>
<td>0001</td>
</tr>
<tr>
<td>a</td>
<td>(not applicable)</td>
<td></td>
<td>10</td>
<td>00</td>
<td>0001</td>
</tr>
<tr>
<td>J</td>
<td>11</td>
<td>1</td>
<td>11</td>
<td>01</td>
<td>0001</td>
</tr>
<tr>
<td>j</td>
<td>(not applicable)</td>
<td></td>
<td>10</td>
<td>01</td>
<td>0001</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>2</td>
<td>11</td>
<td>10</td>
<td>0010</td>
</tr>
<tr>
<td>s</td>
<td>(not applicable)</td>
<td></td>
<td>10</td>
<td>10</td>
<td>0010</td>
</tr>
<tr>
<td>g</td>
<td>(not used), 9</td>
<td></td>
<td>11</td>
<td>11</td>
<td>1001</td>
</tr>
</tbody>
</table>

Note that the "on" and "off" condition of the zone bits in EBCDIC is the reverse of the "on" and "off" condition of the zone bits in BCDIC.

Indicate the following in EBCDIC.

a. Capital "B"
b. Small "b"
ANS:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8421</td>
<td>8421</td>
<td></td>
</tr>
<tr>
<td>a.</td>
<td>1100</td>
<td>0010</td>
</tr>
<tr>
<td>b.</td>
<td>1000</td>
<td>0010</td>
</tr>
</tbody>
</table>

Indicate the following in EBCDIC.

a. Capital "M"

b. Small "m"
ANS:

<table>
<thead>
<tr>
<th></th>
<th>8421</th>
<th>8421</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>1101</td>
<td>0100</td>
</tr>
<tr>
<td>b.</td>
<td>1001</td>
<td>0100</td>
</tr>
</tbody>
</table>

Indicate following in EBCDIC.

- a. Capital "Y"
- b. Number "7"
ANS:  
\[ \begin{array}{ll} 
\text{a.} & 1110 \quad 0101 \quad = "V" \quad \ast \\
\text{b.} & 1111 \quad 0111 \quad = "7" \\
\end{array} \]

*Remember that "V" is 0 zone punch, 5 digit punch.

---

**TEST**

Indicate character in EBCDIC.

<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>8421</th>
<th>8421</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. 4</td>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>b. D</td>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>c. t (small)</td>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>d. 4</td>
<td>.....</td>
<td>.....</td>
</tr>
</tbody>
</table>

---
Special characters used in EBCDIC will not be discussed in this Programmed Instruction. IBM System 360 Reference Data Card provides the bit configuration of special characters in EBCDIC.

A parity check can also be made in EBCDIC by providing an additional check bit. With the exception that there are more bits in EBCDIC than in BCDIC, the method of checking for parity is the same.
The smallest unit of data in a computer is a Bit. Eight bits form the next smallest unit of data, a Byte. In EBCDIC, you've seen that we can use one byte to represent either an alphabetic character, one hexadecimal number, or one special character.

Examples:

<table>
<thead>
<tr>
<th>BYTE ZONE DIGIT</th>
<th>BYTE ZONE DIGIT</th>
<th>BYTE ZONE DIGIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111001</td>
<td>11000001</td>
<td>01011011</td>
</tr>
<tr>
<td>9</td>
<td>A</td>
<td>$</td>
</tr>
</tbody>
</table>

For representing only numeric data, EBCDIC is redundant and wasteful of storage space in the computer. This is because for numeric data in EBCDIC, the zone half of each byte is a hexadecimal F (1111). The major exception to this is the usage of plus (+) and minus (-) signs, which are shown in the zone portion of the right most byte of an EBCDIC number. To show a positive number the F is changed to a C, and to show a negative number the F is changed to a D.

Examples:

F7F1F6F9 = 7169 (This is assumed positive, although not shown)

F7F1F6C9 = +7169

F7F1F6D9 = -7169

Convert the following numbers to EBCDIC format.

a. 943
b. -943
c. +349
d. -8767
As you can see, the F zone is repetitive and wastes storage space. To eliminate this waste, the computer provides the "packed Decimal" or "Binary Coded Decimal" representation. Once again "C" indicates a POSITIVE (+) number, and the "D", a NEGATIVE (-) number. The difference is that each byte contains 2 decimal digits, with the sign shown in the "DIGIT" portion of the rightmost byte in the number.

Example:

Decimal number      + 63264

Represented in:

EBCDIC             F8 F3 F2 F6 C4

Packed Decimal     6 3 2 6 4 C

Note that instead of using 5 bytes, we are now able to use only 3 bytes, thus a saving of 2 bytes.

If the number you are packing does not completely fill the area you've allocated for it, zero's (0) are used as fill on the left.

Example:

F8 F3 F2 D1   EBCDIC

-8321        0 8 3 2 1 D

Indicate the decimal number +421 in EBCDIC format and in packed format.
In "Packing", the proper method for calculating how many bytes will be necessary to accommodate a number is to first count the digits in the number. If it is an even number, add two (2) to it. If it's odd, add one (1). Then divide this number by two. The result will be the number of bytes needed.

Example:

<table>
<thead>
<tr>
<th>Number</th>
<th>Digits</th>
<th>Add</th>
<th>Divide by 2</th>
<th># of bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>+459</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>+6324</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

How many bytes will be needed to pack the number?

a. 8762391
b. 67821467821431
You have seen now how alphabetic characters, number, and special symbols are represented in EBCDIC. You have further seen how we can save storage space by using the "Packed Decimal" feature in representing numbers. Proper knowledge of this is necessary in gaining a complete understanding of computer systems.
SPECIAL TEXT

ST 18-150

DRAFT

BASIC COBOL PROGRAMING

Prepared By
UNITED STATES ARMY INSTITUTE OF ADMINISTRATION
Fort Benjamin Harrison, Indiana — 46216
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>3</td>
</tr>
<tr>
<td>COBOL LANGUAGE CONSTRUCTION</td>
<td>3</td>
</tr>
<tr>
<td>COBOL CHARACTER SET</td>
<td>3</td>
</tr>
<tr>
<td>CHARACTERS USED IN WORDS</td>
<td>4</td>
</tr>
<tr>
<td>PUNCTUATION RULES</td>
<td>4</td>
</tr>
<tr>
<td>CHARACTERS USED IN ARITHMETIC EXPRESSIONS</td>
<td>5</td>
</tr>
<tr>
<td>MARGINS AND THE COBOL CODING SHEET</td>
<td>5</td>
</tr>
<tr>
<td>WORD</td>
<td>5</td>
</tr>
<tr>
<td>COBOL WORDS</td>
<td>6</td>
</tr>
<tr>
<td>THE COBOL LANGUAGE STRUCTURE</td>
<td>8</td>
</tr>
<tr>
<td>IDENTIFICATION DIVISION</td>
<td>9</td>
</tr>
<tr>
<td>PROGRAM-ID</td>
<td>9</td>
</tr>
<tr>
<td>AUTHOR</td>
<td>9</td>
</tr>
<tr>
<td>DATE-COMPILED</td>
<td>9</td>
</tr>
<tr>
<td>REMARKS</td>
<td>9</td>
</tr>
<tr>
<td>ENVIRONMENT DIVISION</td>
<td>11</td>
</tr>
<tr>
<td>SPECIAL-NAMES</td>
<td>12</td>
</tr>
<tr>
<td>SELECT</td>
<td>13</td>
</tr>
<tr>
<td>DATA DIVISION</td>
<td>17</td>
</tr>
<tr>
<td>FD</td>
<td>17</td>
</tr>
<tr>
<td>RECORD DESCRIPTION</td>
<td>21</td>
</tr>
<tr>
<td>WORKING-STORAGE SECTION</td>
<td>24</td>
</tr>
<tr>
<td>77 LEVEL</td>
<td>24</td>
</tr>
<tr>
<td>USAGE</td>
<td>24</td>
</tr>
<tr>
<td>VALUE</td>
<td>24</td>
</tr>
<tr>
<td>SYNCHRONIZED</td>
<td>24</td>
</tr>
<tr>
<td>JUSTIFIED</td>
<td>26</td>
</tr>
<tr>
<td>COMPUTATIONAL</td>
<td>26</td>
</tr>
<tr>
<td>EDITED FIELDS</td>
<td>29</td>
</tr>
<tr>
<td>ARRAYS</td>
<td>31</td>
</tr>
<tr>
<td>REDEFINED</td>
<td>32</td>
</tr>
<tr>
<td>88 LEVEL</td>
<td>33</td>
</tr>
<tr>
<td>PROCEDURE DIVISION</td>
<td>34</td>
</tr>
<tr>
<td>PROCESS VERBS</td>
<td>35</td>
</tr>
<tr>
<td>MOVE</td>
<td>35</td>
</tr>
<tr>
<td>ADD</td>
<td>40</td>
</tr>
<tr>
<td>SUBTRACT</td>
<td>42</td>
</tr>
<tr>
<td>MULTIPLY</td>
<td>43</td>
</tr>
<tr>
<td>DIVIDE</td>
<td>44</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>45</td>
</tr>
</tbody>
</table>

DRAFT ST 18-150
This manual is designed as a student introductory text to 'ANSI' COBOL. Since this is an introductory text, information contained herein may contain oversimplified explanations of the COBOL syntax. More explicit information may be obtained from IBM System/360 Disk Operating System American National Standard COBOL, File Number S360-24, GC28-6394 or IBM System/360 Operating System American National Standard COBOL GC28-6396 (IBM Systems Reference Library, DOS). Information contained herein is extracted mostly from these publications. Army standards were taken from U.S. Army Computer Systems Command Manual 18-1-1, dated 15 May 1975.

The following extract from Government Printing Office Form Number 1965-0795689 is presented for the information and guidance of the user:

"Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention 'COBOL' in acknowledgement of the source, but need not quote this entire section."

"COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations." "No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee in connection therewith." Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages." "The authors and copyright holders of the copyrighted material used herein have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications."

Information in this text represents the language promulgated by the Conference On Data Systems Languages (CODASYL) (the USA standard of the language is American National Standard COBOL, X3.23-1968 (formerly known as USA Standard COBOL, as approved by the American National Standards Institute (ANSI), unless
specifically stated otherwise. COBOL is a problem oriented language which is designed to be highly self-documenting. The structure of the language helps document the program as it is written.
COBOL LANGUAGE CONSTRUCTION. COBOL is based on English. It uses English-type words and certain syntax rules derived from English. However, because it is a computer language, it is much more precise than English. The programmer must, therefore, learn the rules that govern COBOL and follow them exactly.

The basic unit of COBOL is the word. This may be either a COBOL reserved word or a programmer-defined word. Reserved words have a specific syntactical meaning to the COBOL compiler and must be spelled exactly as shown. Programmer-defined words are assigned by the user to such items as data-names (a name made up by the programmer to refer to a specific piece of data in memory). Reserved words and programmer-defined words are combined by the programmer into clauses and statements.

Clauses and statements must be formed following the specific syntactical rules of COBOL. A clause or a statement specifies only one action to be performed, one condition to be analyzed, or one description of data. Clauses and statements can be combined into sentences.

Sentences may be simple (one statement or combination of clauses). Sentences can be combined into paragraphs, which are named units of logically related sentences, and paragraphs can be further combined into named sections.

Both paragraphs and sections can be referred to as procedures, and their names can be referred to as procedure names.

Procedures (sections and paragraphs) are combined into divisions. Divisions are joined into a COBOL program. Each program has exactly four divisions.

COBOL CHARACTER SET. The following symbols may be used in writing COBOL:

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MEANING</th>
<th>SYMBOL</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø,1,...9</td>
<td>digits</td>
<td>,</td>
<td>comma</td>
</tr>
<tr>
<td>A,B,...Z</td>
<td>letters</td>
<td>;</td>
<td>semicolon</td>
</tr>
<tr>
<td>+</td>
<td>plus sign</td>
<td>.</td>
<td>period</td>
</tr>
<tr>
<td>-</td>
<td>minus/hyphen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>asterisk</td>
<td>(</td>
<td>left parenthesis</td>
</tr>
<tr>
<td>/</td>
<td>slash</td>
<td>)</td>
<td>right parenthesis</td>
</tr>
<tr>
<td>=</td>
<td>equal sign</td>
<td>&gt;</td>
<td>greater than symbol</td>
</tr>
<tr>
<td>$</td>
<td>currency sign</td>
<td>&lt;</td>
<td>less than symbol</td>
</tr>
<tr>
<td></td>
<td>space (no printed character)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Characters Used in Words

The characters used in words in a COBOL source program are:
- 0 thru 9
- A thru Z
- (hyphen)

A word is composed of a combination of not more than 30 characters chosen from the character set for words. The word cannot begin or end with a hyphen.

The following characters are used for punctuation:

- period
- right parenthesis
- left parenthesis

PUNCTUATION RULES

The following general rules of punctuation apply in writing COBOL source programs:

1. A period, semicolon, or comma, when used, must not be preceded by a space, but must be followed by a space.

2. A left parenthesis must not be followed immediately by a space; a right parenthesis must not be preceded by a space.

3. At least one space must appear between two successive words and/or parenthetical expressions and/or literals. Two or more successive spaces are treated as a single space, except within non-numeric literals.

4. An arithmetic operator or an equal sign must always be preceded by a space and followed by a space.

5. A comma may be used as a separator between successive operands of a statement. An operand of a statement is shown in a format as a lower case word.

6. A comma or semicolon may be used to separate a series of clauses and is used for ease of reading as in written English.

7. A semicolon may be used to separate a series of statements. Example: ADD A TO B; SUBTRACT B FROM C.
Characters Used in Arithmetic Expressions:

+  addition
-  subtraction
*  multiplication
/  division
** exponentiation
(  left parenthesis
)  right parenthesis
=  replacement operator

NOTE: When any punctuation mark is indicated in a format (in this publication), it is required in the program.

MARGINS AND THE COBOL CODING SHEET

When writing a COBOL program, the programmer is concerned with 2 margins: margin A and margin B. Margin A encompasses columns 8 through 11 of a coding sheet. Margin B encompasses columns 12 through 72.

Margin A is used to identify the different divisions, paragraphs, sections, or special items of a COBOL program. Entries in Margin A may continue through column 72, if necessary, and are terminated by a period. Each statement or clause should be on a single line (card). Margin A statements do not have to begin in column 8, but must begin before column 12.

Army Standard: All entries that can begin in Margin A must begin in column 8.

Margin B contains all the information the programmer wishes to include within a defined division, paragraph, section, or special items of the program. Margin B statements do not have to begin in column 12, but may be written anywhere from column 12 through column 72 (see below).

EXAMPLE:

SPECIAL-NAMES.
   C01 IS CHAN 1.

The word SPECIAL-NAMES begins in margin A.
The words C01 IS CHAN 1 begins in margin B.

WORD

A WORD is composed of not more than 30 characters from the COBOL character set for word formation. The space (blank) is not an allowable character in a word; the space is a word separator. A word is terminated by a space, a period, a right parenthesis, a comma, or a semicolon.
Throughout this publication, basic formats are prescribed for various elements of COBOL. These generalized descriptions are intended to guide the programmer in writing his own statements. They are presented in a uniform system of notation, explained in the following paragraphs. Although it is not part of COBOL, this notation is useful in describing COBOL.

COBOL WORDS

1. All words printed entirely in capital letters are reserved words. These words have preassigned meanings in COBOL. In all formats, words in capital letters represent an actual occurrence of those words. If any such word is incorrectly spelled, it will not be recognized as a reserved word and will probably cause an error in the program. Reserved words cannot be used as procedure or data names.

2. All underlined reserved words are required unless the portion of the format containing them is itself optional. These are key words. If any such word is missing or is incorrectly spelled, it is considered an error in the program. Reserved words not underlined may be included or omitted at the option of the programmer. These words are used only for the sake of readability; they are called optional words, and when used, must be correctly spelled.

3. The characters + - = when appearing in formats, although not underlined, are required when such formats are used.

4. All punctuation and other special characters (except those symbols cited in the following paragraphs) represent the actual occurrence of those characters. Punctuation is essential where it is shown. Additional punctuation can be inserted according to the rules for punctuation specified in this publication.

5. Words that are printed in lower-case letters represent information to be supplied by the programmer. No entry in lower case letters may be omitted, unless it is part of an optional section or phrase which is omitted.

6. In order to facilitate references to them in text, some lower-case words are followed by a hyphen and a digit or letter. This modification does not change the syntactical definition of the word.

7. Square brackets ([]) are used to indicate that the enclosed item may be used or omitted depending on the requirements of the particular program. When two or more items are stacked within brackets, one or none of them may occur.

8. Braces ({} ) enclosing vertically stacked items indicate that one of the enclosed items is obligatory.
9. The ellipsis (...) indicates that the immediately preceding unit may occur once, or any number of times in succession. A unit means either a single lower-case word, or a group of lower-case words and one or more reserved words enclosed in brackets or braces. If a term is enclosed in brackets or braces, the entire unit of which it is a part must be repeated when repetition is specified.

10. Programer defined words are names chosen by the programer to identify program-unique things such as file names or data names. They should be descriptive and clear to someone not familiar with the program.

Army Standard: Only Army standard abbreviations in AR 18-12 may be used in programer defined words.
There are four divisions in each COBOL program. Each is placed in its logical sequence, each has its necessary logical function in the program, and each uses information developed in the divisions preceding it. The four divisions and their sequences are:

IDENTIFICATION DIVISION.
ENVIORNMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
IDENTIFICATION DIVISION

The purpose of the IDENTIFICATION DIVISION is to give the who-what-when-why-where-how of the program to a maintenance programmer. It also identifies the program to the operating system.

BASIC FORMAT:

IDENTIFICATION DIVISION.
  PROGRAM-ID. program-name.
[AUTHOR. your name.]
[INSTALLATION. where the program is written.]
[DATE-WRITTEN. the date that coding began.]
[DATE-Compiled. this entry is completed by the compiler.]
[SECURITY. the classification of the program.]
[REMARKS. a brief description of what the program accomplishes.]

The first words of any COBOL program are IDENTIFICATION DIVISION.

IDENTIFICATION DIVISION must begin in margin A:

IDENTIFICATION DIVISION.

Note that there is one space between IDENTIFICATION and DIVISION, and that IDENTIFICATION DIVISION terminates with a period.

The only required part of the IDENTIFICATION DIVISION is the PROGRAM-ID clause. Since "program-name" is in lower case letters, the programmer must make up a descriptive name by which his program may be identified. All other IDENTIFICATION DIVISION entries help to describe and document your program.

IDENTIFICATION DIVISION.
  PROGRAM-ID. SAMPLE-1.

Note that PROGRAM-ID is written in margin A and follows IDENTIFICATION DIVISION which is written in margin A. After PROGRAM-ID, there is at least one space (a period must be followed by one or more spaces). Then the programmer writes the name of his program, in this case, SAMPLE-1. The program name may be placed anywhere in margin B. An alternate example, which means exactly the same thing as the first example, is listed below.

EXAMPLE:

IDENTIFICATION DIVISION.
  PROGRAM-ID. SAMPLE-1.

All other clauses of the IDENTIFICATION DIVISION are optional. They may be included or deleted as desired. Each clause begins with the word AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPiled, SECURITY, or REMARKS. Each is followed by a period, followed by any comment-type entry, then terminated by a period. This entry, called a sentence, may be repeated as many times as desired.
EXAMPLE:

IDENTIFICATION DIVISION.
PROGRAM-ID. PILPMGT
AUTHOR. LT HUDGIN, ATSG-D-S:
INSTALLATION. USAIA. FORT HARRISON, IN 46216.
DATE-WRITTEN. 14 AUGUST 1976.
DATE-COMPILED. TODAY
SECURITY. UNCLASSIFIED.
*REMARKS. THIS IS A SAMPLE OF A COMPLETE IDENTIFICATION
* DIVISION. ALL OF THE ENTRIES ARE INCLUDED IN THIS EXAMPLE,
* ALTHOUGH ONLY THE PROGRAM-ID PARAGRAPH (WHICH INCLUDES THE
* PROGRAM-NAME) IS REQUIRED.

Army Standards: Although most entries are not required by ANS COBOL, it is a
good practice to include them for documentation purposes.

PROGRAM-ID: The PROGRAM-ID will consist of six positions constructed as
follows:
Position  Entry
1  a. For a program that is operational, this position will
    contain a 'P'.
    b. For a program that is being tested, this position will
       have a 'T'.
    c. If the program's under development, enter a 'D' in this
       position.
2-3  These two positions identify the specific program within a
     system or subsystem.
4-6  The subsystem's or system's identification code.

AUTHOR: Enter the programmer's name and/or the office symbol responsible for
the program.

DATE-COMPILED: Some compilers will not properly place the compilation date
if entry is blank. To avoid this, enter 'TODAY' in this area.

REMARKS: Some compilers do not support this paragraph. To avoid errors,
enter an asterisk (*) in column 7 of every card in this paragraph. This
paragraph should give the reason the program was written, basically how it
works, any special processing or techniques used, what all 77 and 01 entries
in the WORKING-STORAGE SECTION are used for, and any communications with other
programs. Each time a change is made, the maintenance programmer should add
an entry giving the date, his name, why the change was made, and what the
change encompassed.
The ENVIRONMENT DIVISION is used for the following purposes:

a. To identify the computer system to be used (e.g., IBM 360-30, UNIVAC 1108, etc.).

b. To identify the peripheral devices to be used (e.g., card readers, printers, tape drives, etc.).

c. To give or assign a logical name to each peripheral device (e.g., SYS006, SYS004, etc.).

d. To identify the files to be used by the program.

e. To define the organization (sequential, random, etc.) of each file.

f. To assign each file to a specific peripheral device.

BASIC FORMAT:

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
[SOURCE-COMPUTER. computer-name.]
[OBJECT-COMPUTER. computer-name.]
[SPECIAL-NAMES. [function-name-1 IS condition-name-1]]...
INPUT-OUTPUT SECTION.
FILE-CONTROL.

(DOS)*

```
SELECT file-name
ASSIGN TO SYSnnn- \{UT UR\} \{2314 1403 2400 2540R 2540P\} - S.
```

(OS)*

```
SELECT file-name
ASSIGN TO \{UT UR\} \[2314 1403 2400 2540R 2540P\] - S - external-name.
```

* See note on next page.
This paragraph by definition is machine and operating system dependent. We have provided examples of its coding for the two most common operating systems in the Army. No attempt is made to limit this instruction only to IBM hardware or software. Each manufacturer has programming manuals available to explain their own implementation of this paragraph. If device independence is desired when using OS, the device model number must be omitted from the system name and the device class must be UT even if the file is on a unit record device when the program is executed.

There are 2 sections of the ENVIRONMENT DIVISION: The CONFIGURATION SECTION and the INPUT-OUTPUT SECTION. The CONFIGURATION SECTION specifies which computer is to be used to compile the program (the SOURCE-COMPUTER), which computer will be used to execute the program once it is compiled (the OBJECT-COMPUTER), and associates names in the COBOL program with special tests on peripheral devices used in the program.

EXAMPLE:

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-360-G30.
OBJECT-COMPUTER. IBM-360-G30.

In this example, both the source and object computers are the same. The descriptions should be of the smallest computer that the program can be compiled or executed on.

SPECIAL-NAMES is part of the CONFIGURATION SECTION and is used if the programmer wishes to skip to a particular line on the printer page (for example, to the top of the next page). The IBM 1403 printer has a carriage control tape with 12 channels, each of which may have a hole punched in it to indicate where to stop the paper if that particular channel is searched. The carriage control tape used for student jobs has a punch indicating the top of page is on channel 1 and the bottom of the page is on channel 09. The programmer must specify a name which will be associated with the hole punched in the channel if he wishes to skip (slew) to that point on the page. The names to be associated with the channels are specified in the SPECIAL-NAMES clause:

Army Standard: Both the SOURCE-COMPUTER and OBJECT-COMPUTER are required entries.

SPECIAL-NAMES.
  Cnn IS mnemonic-name(...).

"C" means channel; nn is a two-digit number indicating which channel of the twelve possible is to be associated with the name. The mnemonic name is the name that the programmer gives to the channel. It will be used in the PROCEDURE DIVISION skipping to the channel is desired.
EXAMPLE:

SPECIAL-NAMES.

CHAN1 IS CHAN1
CHAN9 IS CHAN9.

Here the word CHAN1 will later be recognized as meaning skip to channel 1 of the carriage control tape on the printer. CHAN9 will mean skip to channel 9.

Army Standards: The only special names that may be used and their purpose are:

<table>
<thead>
<tr>
<th>Special Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAN1</td>
<td>Indicates the first print line page (Channel 1).</td>
</tr>
<tr>
<td>CHAN9</td>
<td>Indicates the last print line of a page (Channel 9).</td>
</tr>
</tbody>
</table>

Channel 1 on the Army standard channel tape is line 6 on the printer page. Channel 9 is line 66. On standard 11" long printer paper printed six lines per inch that leaves one inch margins at the top and bottom.

The FILE-CONTROL paragraph of the INPUT-OUTPUT SECTION is used to tell the system how many files will be used, what type of peripheral devices the files are on, the names to be used to refer to each file, the organization of the files, and where the files are located. Each file used is specified at least once by the use of a SELECT clause and its parts.

To tell the system that a file will be used, write the word SELECT (following the INPUT-OUTPUT SECTION and FILE-CONTROL cards) in margin B. Then devise a name for the file. This name is written after SELECT.

EXAMPLE 1:

INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT MASTER-FILE

EXAMPLE 2:

INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT INVENTORY

In the above 2 examples, the programmer made up the word MASTER-FILE in the first example, and the word INVENTORY in the second. From this point on, these words must be used whenever the programmer wishes to do anything to the file (e.g., READ or OPEN). The name chosen for the file cannot be a COBOL reserved word.
After the file name, the programmer writes ASSIGN TO followed by a system name. The system name can have two forms: one for DOS and one for OS. A DOS system name for a file consists of the logical unit assignment (system logical numbers), the class of device, the model number of the specific device, and the file organization. An OS system name for a file consists of the class, the file organization, and the external name.

In DOS, the system logical number refers to a specific hardware device assigned to that number by JCL (Job Control Language). Its value is a number from SYS000 to SYS243, depending on the parameters chosen during system generation. DOS and OS JCL are discussed later in this text. In OS, the external name performs the same function as the system logical number in DOS. It is limited to from 1 to 8 characters, must begin with a letter, and contain no special characters.

Class refers to the device's capability to handle one or more than one record length. A punched card, for example, is always the same length; it has 80 characters represented on it at all times. UNIT-RECORD (UR) is the class used to describe this type device. Even though no punches are punched in the card, there are still 80 characters there; the space is a character to the computer. When read, therefore, there must be sufficient memory allocated to hold 80 characters, whether they are spaces or any of the other permissible characters of the HOLLERITH or EBCDIC code. The same is true when punching a card, the card punch must know what character is to be punched in each of the 80 columns. A blank card to be punched means, in effect, punch no holes. A similar concept holds true for the printer which must know what character is to be printed in each of 132 print positions. These devices, the card reader, card punch, and printer, are known as UNIT-RECORD devices. They always handle fixed-length records. On the other hand, magnetic tape and disk units are not limited to just one record length. These devices conform to the record length specified for each particular job or file, and when accessing information sequentially, they are referred to as UTILITY (UT) devices.

The letters UR are used to specify UNIT-RECORD; the letters UT are used to specify UTILITY.

After the class is specified, the programmer specifies the model number of the device to be used for the file. This is the hardware model number of the peripherals in the machine configuration at the installation.

For an IBM 360-30 or 360-40, which are the most common Army configurations,

The card reader is specified by 2540R.
The card punch is specified by 2540P.
The disk is specified by 2314.
The printer is specified by 1403.
The magnetic tape is specified by 2400.

The file organization will always be sequential for UR and UT devices.
Sequential organization is specified by S.
All of the parts in the system name are separated by hyphens.

EXAMPLE-1 (DOS). A card file (input):

INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT MASTER-INVENTORY-FILE
       ASSIGN TO SYS006-UR-2540R-S.

EXAMPLE-1 (OS).

INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT MASTER-INVENTORY-FILE
       ASSIGN TO UT-S-MASTER.

The DOS example shows that the name of this file is MASTER-INVENTORY-FILE. The system logical number is SYS006; the card reader is a unit-record device, thus UR; the device number for the card reader is 2540R; and S indicates that it is a SEQUENTIAL file.

The OS example has the same file name, but the device model number is omitted, the class is UT, the organization is sequential, and the external name is MASTER.

EXAMPLE-2 (DOS). An output file on a printer:

INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT LISTING
       ASSIGN TO SYS005-UR-1403-S.

EXAMPLE-2 (OS).

INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT LISTING
       ASSIGN TO UT-S-PRINTER.

The DOS example shows that the name of the file is LISTING; the system logical number is SYS005; UR for UNIT-RECORD; 1403 which is the device number for the printer; and S for SEQUENTIAL file.

The OS example has the same file name, but the device model number is omitted, the class is UT, the organization is sequential, and the external name is PRINTER.

If a program required 2 files, one for input and the other for output, 2 select statements would be required.
EXAMPLE-3 (DOS). A program requiring 2 files--input from disk and output on a printer:

INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT INPUT-FILE
    ASSIGN TO SYS006-UT-2314-S.
SELECT OUTPUT-FILE
    ASSIGN IO SYS005-UR-1403-S.

EXAMPLE-3 (OS).

INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT INPUT-FILE
    ASSIGN TO UT-S-DISK FILE
SELECT OUTPUT-FILE
    ASSIGN TO UT-S-PRINTER.

Note that there is one select statement for each file in the program. The outputs file are similar to the ones in Example 2. For the input file:

**DOS**
The name of the file is INPUT-FILE.
The system logical number is SYS006.
UT for a UTILITY device.
2314 which specifies the type of disk drive on our system.
S for SEQUENTIAL file.

**OS**
The name of the file is INPUT-FILE.
The device class is UT.
The organization is sequential.
The external name is DISK FILE.

For the output file:

**DOS**
The name of the file is OUTPUT-FILE.
The system logical name is SYS005.
It is a UTILITY device (UT).
The device is an IBM 1403 printer.
It is a sequential file (S).

**OS**
The name of the file is OUTPUT-FILE.
The device class is UT.
The organization is sequential.
The external name is PRINTER.
The purpose of the DATA DIVISION is:

1. To describe each file in detail.
2. Allocate memory required by the record of each file.
3. Allocate all other memory needed by the program for working areas (scratch pad).

This text explains the DATA DIVISION in 3 parts: the FILE SECTION FD (file description), the FILE SECTION record description, and the WORKING-STORAGE SECTION.

DATA DIVISION.
FILE SECTION.
FD file-name

[BLOCK CONTAINS integer [CHARACTERS]]
[RECORD CONTAINS integer-l [TO integer-2 CHARACTERS]]

LABEL {RECORD IS} {RECORDS ARE} {OMMITTED STANDARD}

[DATA {RECORD IS} {RECORDS ARE} data-name-1 [data-name-2] ...]

Army Standard: When more than one clause is used to describe a file, each clause must begin on a new card beginning in column 12.

The first part of the DATA DIVISION which is described is the File Description. FD in margin A identifies that what follows is a description of one of the files identified by the SELECT clause in the ENVIRONMENT DIVISION. After FD, place the name of the file to be described in margin B, followed by one or more of the clauses shown above, as applicable. The name of the file must be identical to that of the SELECT clause.

The only required clause is the LABEL clause. All others are optional. It is recommended that the other clauses are used to increase the efficiency of the compiler and for documentation.

All disk files have standard labels. All unit record files have omitted labels.

Army Standard: All possible entries are given for all file descriptions. All tape files must have standard labels unless created on another installation and used as input.
Under OS with no device model number specified in the SELECT statement, the label records are always standard regardless of the device to which the file is ultimately assigned. When used for magnetic tape, labels may appear at the beginning and the end of the file and are used to prevent the wrong program from accessing the information on the tape. When used for disk, labels are used to locate the beginning and end of sequential files. Since the system must know where a sequential file begins and ends on disk, labels must be used on disk; however, they are not mandatory for tape files. It is good programming practice to always use labeled files. This provides another check for the validity of the input or output data.

**EXAMPLE (DOS & OS):**

FILE SECTION.
FD MASTER-IN
LABEL RECORD IS STANDARD.

In this case, MASTER-IN was the name assigned to a file in a SELECT statement in the ENVIRONMENT DIVISION. This file was assigned to either a magnetic tape file or a disk file, if under DOS. All files have standard labels under OS. The LABEL RECORD IS STANDARD, which means to the system, generate standard labels according to my JCL instructions for output files or use the existing standard labels for input files.

If the file called MASTER-IN was assigned to a card reader, the FD would appear as:

**EXAMPLE (DOS):**

FILE SECTION.
FD MASTER-IN
LABEL RECORD IS OMITTED.

**EXAMPLE (OS):**

FILE SECTION.
FD MASTER-IN
LABEL RECORD IS STANDARD.

Since LABEL RECORDS can be used only for magnetic tape and disk under DOS, they must be OMITTED for card readers, punches, and printers. They may also be OMITTED under DOS for magnetic tape if a non-labeled tape is being used. OS always uses standard labels with device independence.

**EXAMPLE (DOS):**

FILE SECTION.
FD PRINTOUT
LABEL RECORD IS OMITTED.
FILE SECTION.
FD PRINTOUT
   LABEL RECORD IS STANDARD.

DOS: In this case, a file called PRINTOUT, assigned to the 1403 printer in the ENVIRONMENT DIVISION, is described in its file description so that LABEL RECORDS ARE OMITTED.

OS: All devices have standard labels in OS regardless of the class.

There are other parts of the FD which may be used to reduce compile time for your program, and for more sophisticated programs, to reduce execution time. (See Appendix C, ANSI COBOL MANUAL.)

The BLOCK CONTAINS clause specifies the blocking factor—that is, how many records are to be read or written at once for each physical input/output operation. The BLOCK CONTAINS must be equal to 1 or omitted for unit-record devices (card readers, card punches, and printers), because a card is physically read, punched or a line is written each time the COBOL program READs or WRITEs a logical record. Physical READs and WRITEs take a lot of time when compared to the internal speed of the computer. So the programer, when using utility devices, can lump together several logical records into one physical record before it is read or written. This cuts down the number of READs and WRITEs on a device and speeds up the execution of the program. The BLOCK CONTAINS clause is needed in this case to tell the compiler that several logical records will be read or written each time the system performs a physical READ or WRITE to or from a device. As far as the programer is concerned, nothing changes in the program except the addition of the BLOCK CONTAINS clause indicating how many of the logical records are to be included in the block. On unit record devices, the logical record must be the same size as the physical record. If BLOCK CONTAINS is omitted, the system defaults to a blocking factor of 1.

EXAMPLE:

FILE SECTION.
FD TAPE-FILE
   BLOCK CONTAINS 3 RECORDS
   LABEL RECORD IS STANDARD.

This entry would not write to the file called TAPE-FILE until 3 records had been built up into a block during processing via WRITE statements. If the file is input, then a block of 3 logical records is read and one is given to the program each time a READ statement for the file is executed.

The RECORDS CONTAINS clause specifies the length of each logical record in the file. A card, for example, contains 80 characters. Thus, the programer may write:
FILE SECTION.
FD TAPE-FILE
  RECORD CONTAINS 80 CHARACTERS
  LABEL RECORDS OMITTED.

Note that, in the BLOCK CONTAINS clause, the programmer may specify the number of characters instead of the number of records. If, on magnetic tape, each record was 40 characters in length, the FD could be written.

FILE SECTION.
FD TAPE-FILE,
  BLOCK CONTAINS 120 CHARACTERS,
  LABEL RECORDS OMITTED.

or

FILE SECTION.
FD TAPE-FILE,
  BLOCK CONTAINS 3 RECORDS,
  LABEL RECORD IS OMITTED.

or

FILE SECTION.
FD TAPE-FILE,
  BLOCK CONTAINS 120 CHARACTERS,
  RECORD CONTAINS 40 CHARACTERS,
  LABEL RECORD IS OMITTED.

NOTE: Commas have no effect on the meaning of anything written. The programmer may use either or both RECORD CONTAINS or BLOCK CONTAINS. Label records may be either STANDARD or OMITTED for tape files, depending on the desires of the programmer. The reason that the RECORD CONTAINS clause may be omitted is that, following the FD clause, COBOL requires that the record for the file be described in detail. The compiler can, therefore, obtain this information from the description of the record (discussed after the FD clause in this text).

The DATA RECORD clause is used to give a name to the record for a file. It is the record which must be placed in memory; therefore, the description of the record is used to allocate storage. To initially specify what the record's name shall be, the programmer uses the DATA RECORD clause.

Army Standard: To avoid duplicate data names, the data names chosen for records descriptions should be prefixed by a four character identifier record and file for that file. It should be in the form AANN where AA is a identifier unique for the file and NN is a two digit number unique for that record in the file. The first character of the file identifier should specify whether the file is input, output, or work ("I", "O", or "W"). The second character should describe the type of file (e.g., "M" for master file, "T" for transactions file, etc.).
EXAMPLE-1:

FILE SECTION.
FD INPUT-FILE
   LABEL RECORD IS OMITTED
   DATA RECORD IS IT01-INPUT-RCD.

EXAMPLE-2:

FILE SECTION.
FD MASTER-INVENTORY-FILE
   LABEL RECORD IS STANDARD
   DATA RECORD IS OM01-MASTER-INVENTORY-RCD.

In the first example, the name of the record for INPUT-FILE is IT01-INPUT-RCD. The prefix means that it is the first ("01") record description of an input ("I") transaction ("T") file. In the second example, the name of the record for MASTER-INVENTORY-FILE is OM01-MASTER-INVENTORY-RCD. The prefix means that it is the first ("01") record description of an output ("O") master ("M") file.

In the entire FD, there is only one period--the one at the end of all entries.

RECORD DESCRIPTIONS

A record is described by placing '01' in margin A, followed by the name of the record in margin B, followed by a description (a PICTURE) of the record.

EXAMPLE:

FILE SECTION.
FD MASTER-FILE
   LABEL RECORD IS STANDARD
   DATA RECORD IS IM01-MASTER-RCD.
01 IM01-MASTER-RCD     PICTURE IS X(100).

In the above example, IM01-MASTER-RCD is the name of the record for MASTER-FILE. The prefix means that it is the first ("01") record description of an input ("I") master ("M") file. Since the label record is STANDARD, this file must either be on a disk or magnetic tape file. The line which begins with 01 (which is called a level number) is used to allocate storage (memory) for the record called IM01-MASTER-RCD. In this case, IM01-MASTER-RCD has a length of 100 (there are 100 characters in the record).

PICTURE CLAUSE.

The PICTURE clause is used to allocate memory. There are 4 types of information which a PICTURE clause describes:
an alphanumeric field, designated by X's.
an alphabetic field, designated by A's.
a numeric field, designated by 9's and optionally a P, S, and/or V.
an edited field, designated by a combination of the characters Z, *, $, +, S, V, B, CR and DB.

Army Standard: (1) The PICTURE clause must be punched beginning in column 48.
(2) PICTURE is preferred abbreviated as PIC.
(3) When more than one clause is used to describe a data item, each clause must begin on a new card beginning in column 40.

An alphanumeric field (X) can contain any legal EBCDIC character. An alphanumeric literal (also called nonnumeric literal) is a character or series of characters enclosed in quotes and is handled as if it were an X (alphanumeric) field. For the continuation of an alphanumeric literal, use all columns through column 72 on the current card, release the card, and punch a dash (-) in column 7 of the next card. The alphanumeric literal is then continued by placing a quote somewhere in margin B and continuing the literal after the quote as though no continuation had been done. It is finished as a normal alphanumeric literal with a quote and period following the end of the alphanumeric literal.

A numeric field (9) is the only field which can be used in mathematical computations or tests and cannot contain any non-numeric characters.

An edit field is designed to edit a numeric or alphanumeric field (e.g., put in dollar signs, decimal points, commas, blanks, etc.)

In the example:

01 OM01-MASTER-RCD PIC X(100).

OM01-MASTER-RCD is described as an alphanumeric field which can hold 100 characters. These characters can be any legal EBCDIC character, including any letter of the alphabet, zero thru nine, blanks, and any special characters such as commas, periods, asterisks, etc.

In order to refer to specific parts of a whole record, the record is "broken down" into smaller parts.

EXAMPLE: A punched card contains the name of an individual in columns 1-20; the individual's age in columns 21-23; spaces in columns 24-80.

To describe this card, the programer breaks the record into parts, using a higher level number to indicate that NAME, for example, is part of CARD-REC. The same holds true for AGE and FILLER. NOTE: Level numbers indicate the hierarchy of data within a record. The range is 01 through 49. Level 01 is highest level and indicates a record in the FILE SECTION. The higher the value of the number, the lower the level indicated. A field of a record may be assigned level number 3. Its subfields may be assigned level number 4.
Level numbers need not be consecutive, but must be sequential. The record level, must always be 01.

Army Standard: Subordinate data items of a record description all will be prefixed by the record prefix. This identifies them uniquely throughout the program. A fifth character may be added between the second and third characters if two files may have the same prefix, e.g., two input master files may be prefixed by IM101 and IM201 respectively.

FILE SECTION.
FD CARD-FILE
  LABEL RECORD IS OMITTED
  DATA RECORD IS IT01-CARD-RCD.
01 IT01-CARD-RCD.
  05 IT01-NAME PIC X(20).
  05 IT01-AGE PIC 999.
  05 FILLER PIC X(57).

In the above example, IT01-NAME, IT01-AGE, and FILLER are all part of IT01-CARD-RCD, because the level number 05 is lower than 01. All numbers except 01, (which is discussed later) appear in margin B. Note that the number in parenthesis (called a duplication factor) indicates the number of times the preceding character is to be repeated. The above example could be written without the duplication factor:

FILE SECTION.
FD CARD-FILE
  LABEL RECORD IS OMITTED
  DATA RECORD IS IT01-CARD-RCD.
01 IT01-CARD-RCD.
  05 IT01-NAME PIC XXXXXXXXXXXXXXXXXXXXX.
  05 IT01-AGE PIC 999.
  05 FILLER PIC XXXXXXXXXXXXXXXXXXXXXXXXXXX.
  05 FILLER PIC XXXXXXXXXXXXXXXXXXXXX.

Notice that the FILLER with a length of 57 characters had to be broken into two pieces. That is, the number of characters used to represent the PICTURE cannot be longer than 30 characters, not the number of characters that PICTURE actually describes.

If a level is to be broken down into smaller parts, it contains no PICTURE clause and is referred to as a group item. The length of a group item is the sum of its subordinate items. To break IT01-NAME down into a first name field and a last name field, the programmer could write:

01 IT01-CARD-RCD.
  05 IT01-NAME.
    10 IT01-FIRST-NAME PIC X(10).
    10 IT01-LAST-NAME PIC X(10).
  05 IT01-AGE PIC 9(3).
  05 FILLER PIC X(57).
IT01-NAME is now a group item and is still equal to a length of 20 characters because its parts' lengths add up to 20. In like manner, IT01-CARD-RCD is equal in length of 80 because its parts' lengths add up to 80.

The DATA RECORD clause need not be specified because the 01 entry is the record name and is implied just by being under its respective FD. It is good programing practice to include it.

Army Standard: As level numbers increase in record descriptions, they will go 01, 05, 10, etc. Columns for each level to begin in are 8, 12, 14, 16, etc., out to column 24. In no case will any level number be punched to the right of column 24.

WORKING-STORAGE SECTION.

The purpose of the FILE SECTION is to describe the files to be used in the program and to allocate memory required for the records of each of the files. The WORKING-STORAGE SECTION is used to allocate all memory needed for holding any data other than records described in the FILE SECTION. All storage in the FILE SECTION is transient. When a record is read, the information that was contained in the record area of the memory before the READ statement is gone forever. When a record is written, the information that was put in the record area of the memory is gone. WORKING-STORAGE, on the other hand, is permanent storage. Variables in this area only change when the program physically moves a new value into them. WORKING-STORAGE should be used for constants; e.g., headers, footers, holding areas, counters, and anything that needs to be kept around during the execution of the program. WORKING-STORAGE is divided into two areas: the 77's area and the 01's area. Each of these will be described in detail later. Because these areas are permanent storage, the compiler will allow you to initialize them to a given value. The record areas in the FILE SECTION cannot be initialized by the compiler since the areas change each time an input or output operation takes place.

Army Standard: All data names in the WORKING-STORAGE SECTION are prefixed by the characters WS-.

EXAMPLE:

WORKING-STORAGE SECTION.
77 WS-LINE-COUNT  PIC S9(4)
               USAGE IS COMP
               VALUE IS ZEROS
               SYNCHRONIZED.

01 WS-HEADING-LINE.
   05 FILLER
   05 FILLER

   PIC X(64)
   VALUE IS SPACES.

   PIC X(69)
   VALUE IS 'TITLE'.
A 77 data item is normally used for counters and hold areas. All 77 items appear immediately after WORKING-STORAGE SECTION and before the first 01 record entry. They are always elementary items and therefore cannot be subdivided with larger numbers like 78. In the example above, a 77 item named WS-LINE-COUNT is defined and initialized to a value of zero. The other area in WORKING-STORAGE is the 01 area. This area contains a series of 01 group or elementary items. The 01 group items may be broken down in a manner similar to the record descriptions in the FILE SECTION. In the example above, a group item named WS-HEADING-LINE is defined then broken down into two FILLERs. The first FILLER simply takes up 64 characters of storage and initializes them to spaces. The second FILLER takes up 69 characters of storage and initializes the first 5 to the characters TITLE. The other 64 characters of the 69 are initialized to spaces. Use of the value clause will be described in detail later. All items written as 77 entries can also be written as 01's. 01's can be elementary or group items.

EXAMPLE:

77 WS-LINE-COUNT PIC S9(4)
    USAGE IS COMP
    VALUE IS ZEROS
    SYNC.

77 WS-PAGE-COUNT PIC S9(4)
    USAGE IS COMP
    VALUE IS ZEROS
    SYNC.

77 WS-PERSON-COUNT PIC S9(4)
    USAGE IS COMP
    VALUE IS ZEROS
    SYNC.

This could be coded as:

01 WS-COUNTERS.
   05 WS-LINE-COUNT PIC S9(4)
       USAGE IS COMP
       VALUE IS ZEROS
       SYNC.

   05 WS-PAGE-COUNT PIC S9(4)
       USAGE IS COMP
       VALUE IS ZEROS
       SYNC.

   05 WS-PERSON-COUNT PIC S9(4)
       USAGE IS COMP
       VALUE IS ZEROS
       SYNC.
Army Standard: The clauses included with a data item description should be in the following order:

- Redefines
- Occurs
- Picture (PIC)
- Usage
- Value
- Justified
- Synchronized (Sync).

All clauses mentioned above will be described later in detail.

All three methods yield the same result as far as defining the data items and making them available for use in the Procedure Division. The difference lies in the resultant storage allocation. It is enough for the beginning programmer to know that for this type of data item the 77 is the most efficient and preferable of the three.

If the programmer needs a numeric field and if that field will only be used for mathematical computation (not part of an input or output record, i.e., line counter), the programmer may specify that the usage of the numeric field is computational. This may be abbreviated as COMP. COMP fields are much more efficient for integer arithmetic than simple PIC 9 fields. You may not put in a V in the picture. To print these fields is very time consuming, therefore, these should only be used for internal program counters. If the programmer chooses to specify computational (for efficiency purposes), he should guarantee that this field begin on a correct boundry (a computational field, on IBM System/360, is a numeric field which should begin either on a half or full word boundry). To direct that this field is to start on a correct boundry, the programmer specifies synchronized (which may be abbreviated Sync). See the above example for this.

Army Standard: synchronized and computational are preferred to be abbreviated.

Note: A numeric field which is computational must include S (sign) as the
leftmost character in the picture clause.

If SYNCHRONIZED is used to describe a field which does not need to be aligned on a boundary (e.g., PICTURE X), it is ignored.
In the following examples, WS-AGE, not being a COMPUTATIONAL field, does not have to be aligned on any boundary; thus SYNCHRONIZED is ignored.

EXAMPLE:

01 WS-CARD-k-D.
   05 WS-NAMF.
      10 WS-LAST-NAME PIC X(10).
      10 WS-FIRST-NAME PIC X(10).
   05 WS-AGE PIC X(3).

05 WS-AGE PIC 999.

If WS-AGE were not to be used as either a mathematical computation in the program or a numeric edited field on output, it could be defined as an alphanumeric (X) field:

05 WS-AGE PIC X(3).

Note that each level entry ends with a period.

Any elementary item in WORKING-STORAGE (item with a PICTURE) may be assigned an initial value via the VALUE IS clause. Group items cannot be assigned a value and any data item under a REDEFINES clause cannot have a VALUE. REDEFINES is discussed in detail later. The only thing to keep in mind when assigning a VALUE to something is to remember how that thing is described in its PICTURE. Numeric items cannot be assigned an alphanumeric VALUE. Likewise, alphanumeric items should not be assigned a numeric literal VALUE; however, this is more acceptable than the first case. For example, one would not want to describe a counter as PIC 999 then try to initialize that counter to spaces. Spaces are not numbers and will not compute. COBOL has made things a little easier for programmers to initialize variables. There exist a number of reserved words that represent specific values when associated with a data item. ZEROS is used with numeric PICTURES. SPACES is used with alphanumeric PICTURES.

EXAMPLES (Not in the order that they would be in a program):

01 WS-BLANK-LINE PIC X(133) VALUE IS SPACES.
01 WS-COUNTER PIC 9(3) VALUE IS ZEROS.
77 WS-LINE-COUNT PIC S9(4) VALUE IS ZEROS.
01 WS-TITLE-1 PIC X(5) VALUE IS 'TITLE'.
77 WS-PAGE-COUNT PIC 999 VALUE IS 0.
The first example defines a 133 character block of memory and initializes it to spaces. The second example defines a 3 character numeric data item and initializes it to zeros. The third example defines a signed numeric variable of 4 characters and initializes it to zeros. The S in front of the 9 indicates that this variable may be negative or positive. The absence of an S indicates that this variable will always be positive and any value that is moved to that item is made positive. The fourth example defines a 5 character alphanumeric item and is initialized to a value of 'TITLE'. This example uses another form of initialization: the literal. Literals can be either alphanumeric (enclosed in quotes) for X PICTUREs or numeric for 9's PICTURES. PICTURES are described in detail in the next section. If a numeric PICTURE has an S and a numeric literal is used, then that numeric literal must have a + or - in front of it.

EXAMPLE:

77 LINE-COUNT  
   PIC S9(4)  
   USAGE IS COMP 
   VALUE IS +100 
   SYNC.

It is important to remember that a literal, whether numeric or alphanumeric, cannot exceed the length of the associated PICTURE. It was mentioned before that a numeric value should not be assigned to an alphanumeric PICTURE. ZEROS cannot be associated with an X PICTURE. However, the following is legal and desirable sometimes:

EXAMPLE:

77 NUMERIC-LITERAL  
    PIC X(10)  
    VALUE IS '0123456789'.

This example defines a 10 character alphanumeric data item and initializes it to the character string 0123456789. Anything placed within the quotes of the VALUE clause in an alphanumeric literal is legal and allowable. They are just another string of characters as far as the compiler is concerned. It should be remembered that one still cannot do arithmetic with the above example because the PICTURE is alphanumeric. If the literal used with an alphanumeric item is shorter than the PICTURE, the literal is placed in the left of the field and padded to the right with spaces.

RESTRICTION OF THE VALUE CLAUSE: IT CAN ONLY BE USED IN THE WORKING-STORAGE SECTION; IT CANNOT BE USED IN THE FILE SECTION.

Numeric literals may contain numeric digits (0 thru 9), a sign if the PICTURE clause has an S and a decimal point to indicate proper decimal point alignment.
EXAMPLES:

77 WS-PAGE-COUNTER              PIC 999
       VALUE IS 1.

77 WS-TAX-RATE                   PIC SV99
       VALUE IS +.18.

77 WS-TOTAL-TAX                  PIC S9(4)V99
       VALUE IS +0.

Note that even when the literal has a decimal point in it, a period still ends the data item description giving two periods in the description.

If the literal used with a numeric PICTURE is shorter than the PICTURE, then the literal is placed in the right of the field and padded to the left with zeros. The only exception to this is when a V is included in a numeric PICTURE.

EXAMPLE:

77 WS-TAX-RATE                   PIC SV99
       VALUE IS +.23.

77 WS-NET-PAY                    PIC S9(5)V99
       VALUE IS +25.

In the first example, TAX-RATE is defined as a two character numeric variable with the implied decimal point to the left of the two digits. There is no actual decimal point in the field, but arithmetic is done as though it existed. Any number moved to this field will be adjusted so the implied decimal point is in the appropriate place. For example, if one moved the integer 33 to WS-TAX-RATE, WS-TAX-RATE would have zeros in it because 33 implied 33.00 and WS-TAX-RATE only has places for the numbers to the right of the decimal point. However, if one moved 33.25 to WS-TAX-RATE, WS-TAX-RATE would contain 25 with the implied decimal point to the left of the numbers. Decimal points are not numbers so they cannot be physically in the field that has the number being used in the computation. The V is used only for adjusting the significance of the digits. If it is possible for a move or computation to result in losing high order digits, a message will be printed in the error listing warning the programmer of the possibility. In the second example, WS-NET-PAY is defined as having seven characters of memory and is initialized to a value of 0012500 with the right two digits implied to be decimals. Notice that after the adjustment was made for the decimal point, the field was padded in both directions with zeros.

EDITED FIELDS

An EDI field is designed to put symbols such as $, ., * + - in the correct place to make a numeric field more understandable. The following symbols may appear in a report field:

<table>
<thead>
<tr>
<th>List 1:</th>
<th>S V 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>List 2:</td>
<td>Z , $ . + - * B CR DB</td>
</tr>
</tbody>
</table>
If one or more symbols from List 2 are used in a PICTURE clause, the field becomes a numeric EDIT field and cannot be used in any mathematical computation. If only the symbols in List 1 are used, the field remains a numeric field and may be used in any mathematical computation. The only way to get information into a EDIT field is to MOVE a numeric field TO an EDIT field (this instruction is given in the PROCEDURE DIVISION). Usually EDIT fields are used as part of an output description for a print file.

**CHARACTER MEANING**

- **Z** Zero suppress non-significant leading zeros.
- **,** Put a comma here if there are any significant digits to the left.
- **$** If one occurrence, put a dollar sign here; if multiple occurrences, put a dollar sign to the left of the first significant digit.
- **.** Decimal point.
- **+** Print a plus sign if the number moved into the field is positive; print a minus sign if the field is negative.
- **-** Print a space if the number moved into the field is positive; print a minus sign if the field is negative.
- ***** Check protection: replace non-significant leading zeros with asterisks.
- **V** Implied decimal position. This does not take up a print position or a column on input.
- **S** Keep track of the sign for numeric fields (do not use S if any character from list 2 appears in the PICTURE clause).
- **B** Generate a blank wherever it is inserted.
- **CR** Prints two spaces if the value of the number moved into the field is positive. Prints as CR if the value is negative.
- **DB** Prints two spaces if the value of the number moved into the field is positive. Prints as DB if the value is negative.
EXAMPLES OF EDITED MOVES:

<table>
<thead>
<tr>
<th>PICTURE</th>
<th>VALUE OF DATA</th>
<th>EDITED RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$$$,.99</td>
<td>V12</td>
<td>$.12</td>
</tr>
<tr>
<td>$$,$$9.99</td>
<td>500V25</td>
<td>$500.25</td>
</tr>
<tr>
<td>$,$,$$9.99</td>
<td>1500V25</td>
<td>$1,500.25</td>
</tr>
<tr>
<td>$Z,ZZ9.99</td>
<td>-V25</td>
<td>$.00</td>
</tr>
<tr>
<td>$Z,ZZ9.99-</td>
<td>$0.25-</td>
<td></td>
</tr>
<tr>
<td>$Z,ZZ9.99-</td>
<td>$1.50</td>
<td></td>
</tr>
<tr>
<td>$*,**9.99-</td>
<td>25V75</td>
<td>$***25.75</td>
</tr>
<tr>
<td>$*,**9.99+</td>
<td>25V75</td>
<td>$***25.75+</td>
</tr>
<tr>
<td>$*,**9.99+</td>
<td>-25V75</td>
<td>$***25.75-</td>
</tr>
<tr>
<td>$*,**9.99BCR</td>
<td>25V75</td>
<td>$***25.75 CR</td>
</tr>
<tr>
<td>$*,**9.99BCR</td>
<td>-25V75</td>
<td>$***25.75 CR</td>
</tr>
<tr>
<td>$*,**9.99CR</td>
<td>-25V75</td>
<td>$***25.75 CR</td>
</tr>
<tr>
<td>$*,**9.99BDB</td>
<td>25V75</td>
<td>$***25.75 DB</td>
</tr>
<tr>
<td>$*,**9.99BDB</td>
<td>-25V75</td>
<td>$***25.75 DB</td>
</tr>
<tr>
<td>$*,**9.99DB</td>
<td>-25V75</td>
<td>$***25.75 DB</td>
</tr>
<tr>
<td>$+,+++99</td>
<td>25V75</td>
<td>+25.75</td>
</tr>
<tr>
<td>$+,+++99</td>
<td>-25V75</td>
<td>-25.75</td>
</tr>
<tr>
<td>$-,---99</td>
<td>25V75</td>
<td>25.75</td>
</tr>
<tr>
<td>$-,---99</td>
<td>-25V75</td>
<td>-25.75</td>
</tr>
</tbody>
</table>

V's are shown for explanation only.

ARRAYS (TABLES)

An array is nothing more than a series of identical data items. Instead of having to define all of them individually, COBOL allows one to say that one thing OCCURS n times. All that the programmer has to do is define what one of the things looks like. The things are then referenced by use of an index or a subscript. The index or subscript simply tells the compiler which of the things you want to look at.

EXAMPLE:

```cobol
01 WS-TABLE.
   05 WS-TABLE-ENTRY OCCURS 25 TIMES
      INDEXED BY TABLE-INDEX.
      10 WS-NAME PIC X(20).
      10 WS-SOC-SEC-NUM PIC X(9).
```

This example defines an array named WS-TABLE made up of 25 WS-TABLE-ENTRY's. Each WS-TABLE-ENTRY is broken down into a WS-NAME and WS-SOC-SEC-NUM. The entire array is 580 characters long. ((20+9) x 25) This array is indexed by WS-TABLE-INDEX. How to reference arrays in the PROCEDURE DIVISION is covered in the section on the PROCEDURE DIVISION.
RESTRICTIONS ON ARRAYS: The OCCURS clause cannot be used with 77 or 01 items. Any item containing an OCCURS clause or subordinate to an item containing an OCCURS clause cannot have a VALUE clause.

REDEFINES CLAUSE

The same area of memory may be described in more than one way by using the REDEFINES clause. Many times it is necessary to use a particular data item in several ways (alphanumeric or numeric). This is done with the REDEFINES clause.

EXAMPLE:

```
01 WS-DISK-RCD.
   05 WS-PAY       PIC S9(5)V99.
   VALUE ZEROS.
   REDEFINES WS-PAY
   PIC X(7).

01 WS-CARD-RCD.
   05 WS-NAME-1.
      10 WS-LAST-NAME-1   PIC X(12).
      10 WS-FIRST-NAME-1  PIC X(8).
   05 WS-NAME-2
      10 WS-FIRST-NAME-2  PIC X(8).
      10 WS-LAST-NAME-2   PIC X(12).
```

In the first example, WS-PAY is simply REDEFINED as alphanumeric by WS-PAY-1. This type of redefinition is particularly useful when checking the validity of numeric fields. More of this will be covered in the discussion of the PROCEDURE DIVISION. In the second example, the storage assigned to WS-NAME-1 is defined in two ways: WS-LAST-NAME then WS-FIRST-NAME and WS-FIRST-NAME then WS-LAST-NAME. When the data coming into the program can be of several different forms, the REDEFINES clause is very helpful.

RESTRICTIONS ON THE REDEFINES CLAUSE:

1. Redefinition within the elements subordinate to an OCCURS clause will generate a warning.
2. Be sure to make the two items (the object and subject of the REDEFINES) the same length.
3. REDEFINES cannot be specified in an 01 entry in the FILE SECTION.
4. If A REDEFINES B, then both A and B must have the same level number and there must be no intervening level numbers having a level number equal to or less than A and B's level numbers. For example, if A and B were both 05
entries, then no entry with 05 or less could be between A and B. In example 2 above, there are 10's between NAME-1 and NAME-2. Both NAME-1 and NAME-2 are 05 entries.

88 LEVEL ITEMS

An 88 level item may be defined under any data item, both elementary and group level. It has no PICTURE clause. Instead it is used to describe possible values or ranges of values and to associate a name with the description.

EXAMPLE:

```plaintext
05 IT01-TRANSACTION-CODE PIC X.
   88 ADDITION VALUE 'A'.
   88 CHANGE VALUE 'C'.
   88 DEF'ETION VALUE 'D'.
```

In this example, the elementary data item IT01-TRANSACTION-CODE is described as having the possible value of A, C, or D. If one of these values occurs, the name associated with them is true. See the discussion of the IF statement to see how to use the 88 in the PROCEDURE DIVISION. If any other value other than the ones described in the 88's occurs, none of the 88's are true, but no error condition is raised.

EXAMPLE:

```plaintext
01 WS-PAY PIC 9(6)V99.
   88 SMALL VALUE .00 THRU 100.00.
   88 MEDIUM VALUE 100.01 THRU 300.00.
   88 LARGE VALUE 300.01 THRU 1000.00.
   88 VERY-LARGE VALUE 1000.01 THRU 999999.99
```

Note in the first example that the values associated with the IT01-TRANSACTION-CODE are enclosed in quotes because the PICTURE of IT01-TRANSACTION-CODE is alphanumeric. In the second example, the VALUES are numeric literals because the PICTURE of WS-PAY is numeric. 88 level entries can be in both the FILE SECTION and the WORKING-STORAGE SECTION. They should not be used under 77 level items. Ranges may be used with alphanumeric items, but care should be exercised.

EXAMPLE:

```plaintext
05 WS-DIVISION-CODE PIC XX.
   88 DIVISION-1 VALUE '01' THRU '02'.
   88 DIVISION-2 VALUE '03' THRU '04'.
```

What is being described is the internal machine representation of '01' thru the internal representation of '02' and all intervening characters. This type of 88 is very machine dependent and can be the cause of obscure errors in processing.
THE PROCEDURE DIVISION

The PROCEDURE DIVISION contains the instructions that manipulate the data described in the DATA DIVISION. The PROCEDURE DIVISION is made up of a series of paragraphs. Each paragraph begins with a name (the paragraph name) followed by a period followed by the body of the paragraph. The body is made up of one or more COBOL PROCEDURE DIVISION sentences. A COBOL sentence is one or more COBOL verbs and their associated parts. A sentence is ended by a period. All paragraph names begin in margin A. All sentences begin in margin B.

Army Standard: All paragraph names will be preceded by a four digit number. Paragraphs will be in ascending order by this number. The numbers will be incremented by 10 to allow insertion of new paragraphs in the future.

EXAMPLE:

PROCEDURE DIVISION.
0010-PARA-NAME.
  OPEN INPUT CARD-READER
  OUTPUT PRINTER.
  PERFORM 0020-READ-AND-PRINT THRU 0020-RAP-EXIT
      UNTIL WS-EOF-SWITCH = 'OFF'.
  CLOSE CARD-READER,
      PRINTER.
  STOP RUN.
0020-READ-AND-PRINT.
  READ CARD-READER
      AT END MOVE 'OFF' TO WS-EOF-SWITCH
      GO TO 0020-RAP-EXIT.
  MOVE CARD-RECORD TO PRINT-RECORD.
  WRITE PRINT-RECORD AFTER ADVANCING 2 LINES.
0020-RAP-EXIT. EXIT.

In the above example, there are three paragraphs. The first starts with 0010-PARA-NAME and ends with STOP RUN. The second paragraph begins with 0020-READ-AND-PRINT and ends with WRITE PRINT-RECORD AFTER ADVANCING 2 LINES. The third begins with 0020-RAP-EXIT and ends with EXIT (this has only one sentence in it). EXIT will be explained later, but for now, the EXIT paragraph is considered to be the end of the preceding paragraph. What this program actually does is immaterial at this point. Each of the verbs (OPEN, PERFORM, CLOSE, STOP, READ, MOVE, GO, WRITE) will be discussed in detail later. When the PROCEDURE DIVISION is executed, the first statement after PROCEDURE DIVISION is executed first. Because of this, no paragraph name is needed immediately after PROCEDURE DIVISION. ANSI requires para name. Normally paragraphs represent blocks of code that do one job. In the example, everything from 0020-READ-AND-PRINT thru 0020-RAP-EXIT is involved in the reading and writing of data records. Everything from the PROCEDURE DIVISION thru the STOP RUN is involved in preparing the files to be read and written, controlling the reading and writing, then cleaning up after it has all been done. Each paragraph has its own job and doesn't depend on anything else to do that job for it.

DRAFT ST 18-150
The PROCEDURE DIVISION's verbs have been divided into three classes:

1. Process verbs - words which initiate manipulation of data in the program.
2. I/O verbs - words which initiate data transfer from the outside world into the program or vice versa.
3. Logic control verbs - words which affect the order of statement execution.

Each of these categories of verbs are discussed below.

PROCESS VERBS

MOVE

The form of the MOVE verb is:

\[
\text{MOVE } \{ \text{data-name-1} \} \text{ TO } \text{data-name-2} \ [\text{data-name-3} \ldots]\]

This verb copies information from one data-name or literal to one or more data-names. There are several restrictions on moving data from one area to another that are connected to the PICTUREs of the respective data-items. See Table 1. An example of the MOVE statement is:

EXAMPLE:

MOVE IM01-NAME TO WS-NAME.

MOVE IM01-DIVISION TO WS-DIVISION
WS-DIVISION HOLD.

In the first example, the data-item IM01-NAME is copied to WS-NAME. When the move is complete, both IM01-NAME and WS-NAME contain the information that IM01-NAME had contained before the MOVE was executed. In the second example, the information in IM01-DIVISION is copied to both WS-DIVISION and WS-DIVISION HOLD. When the MOVE is complete, all three data items have the same information in them.

Another form of the MOVE statement is:

\[
\text{MOVE } \{ \text{CORRESPONDING/ group-item-1,} \text{ TO group-item-2.} \}
\]

This form of the MOVE verb copies all of the data-names under group-item-1 that have the same data-names under group-item-2. When the MOVE CORRESPONDING is complete, anything from all of the data-items under group-item-1 to none of the data-items under group-item-1 may have been moved.
Army Standard: Since non-unique names are not allowed and the move corresponding depends on non-unique names, the move corresponding statement is not allowed.

EXAMPLE:

FILE SECTION.

\[1\] IM01-MASTER-RCD.
\[5\] IM01-NAME  PIC X(20).
\[5\] IM01-SSN    PIC X(9).
\[5\] IM01-RANK  PIC X(3).
\[5\] FILLER    PIC X(48).

WORKING-STORAGE SECTION.

\[1\] WS-DETAIL-LINE-1.
\[5\] FILLER   PIC X(48).
\[5\] VALUE IS SPACES.
\[5\] WS-NAME  PIC X(20).
\[5\] FILLER   PIC X(3).
\[5\] VALUE IS SPACES.
\[5\] WS-RANK  PIC X(3).
\[5\] FILLER   PIC X(3).
\[5\] VALUE IS SPACES.
\[5\] WS-SSN   PIC X(9).
\[5\] FILLER   PIC X(47).
\[5\] VALUE IS SPACES.

\[1\] WS-DETAIL-LINE-2.
\[5\] FILLER   PIC X(48).
\[5\] VALUE IS SPACES.
\[5\] IM01-NAME PIC X(23).
\[5\] IM01-RANK PIC X(6).
\[5\] IM01-SSN PIC X(56).

PROCEDURE DIVISION.

MOVE CORRESPONDING IM01-MASTER-RCD TO WS-DETAIL-LINE-1.
MOVE CORRESPONDING IM01-MASTER-RCD TO WS-DETAIL-LINE-2.

In the first MOVE CORRESPONDING statement, nothing would have been moved and no error message generated. This is one weakness of the MOVE CORRESPONDING statement. None of the data-items are named identically under group-item-1 and group-item-2. In the second MOVE CORRESPONDING, all of the data-item in MASTER-RECORD are moved in DETAIL-LINE-2. Note that instead of defining all of the FILLERs between the data-items as in DETAIL-LINE-1, the data-items in DETAIL-LINE-2 are simply defined as containing those FILLERs. When the
IM01-NAME defined in IM01-MASTER-RECORD is moved to the IM01-NAME defined in WS-DETAIL-LINE-2, the information is moved into the left most of the 23 character receiving field and the remaining 3 characters are filled with blanks. Similar actions accompany the other two moves. This type of definition is less efficient than the first because the trailing characters in all of the receiving fields are filled with blanks each time the move statement is executed. In the first example, the blanks are put in only once during the compilation and remain there all during the execution of the program with no modification. It is normally a good idea to make receiving fields the same size as sending fields.

NOTE: If any of the data item names are defined with the same data name in two or more areas and need to be referenced individually, they may be referenced by qualifying the item with the qualifier OF.

EXAMPLE:

MOVE RANK OF MASTER-RECORD TO RANK-HOLD.

Army Standard: Qualification of data names will not be used.

TABLE 1

LEGAL MOVES:

1. (A) an alphabetic field may be moved to another alphabetic field. The movement is from the leftmost byte of the receiving field to the right. If the receiving field is larger, the remaining positions will be filled with spaces. If the receiving field is shorter than the sending field, truncation of the right most position(s) will occur.

2. (X) an alphanumeric field may be moved to another alphanumeric field. The same rules apply here as they do in an alphabetic field.

3. (9) a numeric field may be moved to another numeric field. However, the movement is accomplished by decimal point alignment. If the receiving field is smaller, truncation will occur and if the receiving field is larger, zeros will fill in the remaining core area. Exactly how the truncation takes place will be explained later.

4. A numeric or alphabetic field may be moved to an alphanumeric field. When numeric (PIC 9's) date fields are moved to alphanumeric (PIC X's) data fields, the move is treated as an alphanumeric move and follows the rules shown under rule 1.

5. An edited field may be moved to an alphanumeric field.

6. A numeric field may be moved to an edited field.

7. An alphabetic field may be moved to an alphanumeric edited field.
EXAMPLE:

MOVE A-FIELD TO B-FIELD.

If B-Field were alphanumeric, A-Field could be alphabetic, alphanumeric or edited.
If B-FIELD were numeric, A-FIELD could only be numeric.
If B-FIELD were alphabetic, A-FIELD could only be alphabetic.
If B-FIELD were edited, A-FIELD could only be numeric.

In the above examples, A-FIELD remains unchanged; B-FIELD becomes a "copy" of A-FIELD.

EXAMPLE:

MOVE SPACES TO OR01-PRINTER-RCD.

In this case, OR01-PRINTER-RCD must be defined as PICTURE X (alphanumeric) or PICTURE A (alphabetic), because spaces is recognized by the compiler as 1 or more "blanks", which is a character, not a number, and can be placed only into a PICTURE (A) or (X) fields.

NOTE: All group level items (items which are broken down into smaller parts) are handled as PICTURE X fields regardless of the PICTURE of the subordinate elements of the field are described.

EXAMPLE:

01 WS-OUI-RCD.
  05 WS-NUMBER PIC 9(6).
  05 FILLER PIC X(127).

In this example, WS-OUI-RCD is alphanumeric (PICTURE (X)) because it is a group item. Its subordinate parts are WS-NUMBER and FILLER, both elementary items because they have PICTURE clauses. WS-OUT-RCD's length is the sum of its parts (133 characters). The programmer can legally write MOVE SPACES TO WS-OUT-RCD because WS-OUT-RCD is a group item and therefore alphanumeric even though the first six positions are defined as numeric. If this move was done, then WS-NUMBER cannot be used in arithmetic computations because the information in it is not numeric and will cause a 'data exception' (trying to use non-numeric data as though it were numeric). Numeric information must be moved to WS-NUMBER before WS-NUMBER can be used as a number.

If the two items are alphanumeric and not the same size, one of two things will happen:

1. If the smaller item is moved to the larger, the data in the smaller item is put as far left as possible in the larger field. The remaining characters of the larger item will be filled with spaces.
2. If the larger item is moved to the smaller, the smaller item is filled up with the information from the larger starting at the left of both fields. The extra characters in the larger field are ignored (truncated).

EXAMPLE:

```plaintext
01 IM01-CARD.
  05 IM01-NAME       PIC X(20).
  05 IM01-SSN        PIC X(9).
  05 IM01-ADDRESS    PIC X(40).
  05 FILLER          PIC X(11).

01 WS-DETAIL-LINE.
  05 FILLER           PIC X(33).
  05 WS-NAME          PIC X(25).
  05 WS-SSN           PIC X(15).
  05 WS-ADDRESS       PIC X(30).
  05 FILLER           PIC X(32).
```

MOVE IM01-NAME TO WS-NAME.
MOVE IM01-SSN TO WS-SSN.
MOVE IM01-ADDRESS TO WS-ADDRESS.

IM01-NAME is moved to the left most twenty characters of WS-NAME and the remaining five characters are filled with spaces. IM01-SSN is likewise put in the leftmost nine characters of WS-SSN and the remaining six characters are filled with spaces. Since IM01-ADDRESS is larger than WS-ADDRESS, only the leftmost thirty characters of IM01-ADDRESS are moved to WS-ADDRESS.

All numeric fields have a decimal point either implied (with a 'V'), assumed (if no other decimal point is given, it defaults to the right of the rightmost digit), or indented (as a '.'). Movement from one numeric field to another takes place after decimal point alignment. One or both of these cases may apply.

1. After decimal point alignment, the digits are moved from the sending field into the receiving field, beginning at the left of the decimal point and proceeding left. If the sending field has more digits to the left of the decimal point than the receiving field, only as many as will fit are moved. The rest are truncated. If the sending field has fewer digits to the left of the decimal point than the receiving field, then the extra digits are filled with zeros.

2. If the sending field has more digits to the right of the decimal point than the receiving field, then only as many as will fit are moved. The rest are truncated. If the sending field has fewer digits to the right of the decimal point than the receiving field, then the extra digits are filled with zeros.
EXAMPLES:

77 WS-NUM-1
    VALUE 25.1.
77 WS-NUM-2
    PIC 999V99.
77 WS-NUM-3
    PIC V999.
77 WS-NUM-4
    PIC 99.

MOVE WS-NUM-1 to WS-NUM-2,
    WS-NUM-3,
    WS-NUM-4.

After the moves, the value of WS-NUM-1 is 25.1, WS-NUM-2 is 025.10, WS-NUM-3 is .100, and WS-NUM-4 is 25.

To the left of the decimal point, the data is right justified. Any excess high order positions are filled with 0's. To the right of the decimal point, they are left justified. Any excess low order positions are filled with 0's. Truncation occurs as follows: To the left of the decimal point, the high order is truncated. To the right of decimal point, the low order is truncated.

ADD

There are three forms of the ADD verb. The first form is normally used to add one or more things to a sum.

```
ADD (data-name-1 numeric-literal-1) (data-name-2 numeric-literal-2) ...
    data-name-m numeric-literal-m
    TO data-name-n [ROUNDED]
[ON SIZE ERROR any series of COBOL verbs and associated parts except IF-ELSE (period)]
```

EXAMPLE 1:

ADD WS-FICA-TAX WS-CITY-TAX WS-LOCAL-TAX WS-STATE-TAX TO WS-TOTAL-TAX.

EXAMPLE 2:

ADD 1 to WS-LINE-COUNT.

The first example adds four individual taxes to the total already in WS-TOTAL-TAX yielding a new total. This list of taxes could also be added to several other things in the same statement by simply listing the things after WS-TOTAL-TAX. Each of the resultant fields may have the ROUNDED clause included after it. If ROUNDED is used, the result is rounded according to the receiving field PICTURE. After decimal point alignment, the number of places to the right of the decimal point of the result of an arithmetic computation is compared with the number of places allowed for the result in the receiving field. If more
places are in the result than can be accommodated for in the receiving field, normally the extra places are simply discarded (truncated). If ROUNDED is specified, then the most significant digit of the excess is examined. If this digit is 5 or more, then the least significant digit of the part that is going to be used will be incremented by one. If this digit is 4 or less, then the number remains the same and the unneeded low order digits are discarded. If ON SIZE ERROR is specified after the list of receiving fields, then if the result of the computation is too large to fit in the specified PICTURE, the list of verbs after the ON SIZE ERROR is executed. Whether the ON SIZE ERROR clause is specified or not and a high order truncation condition occurs, the truncated result is still put in the receiving fields. In the second example, WS-LINE-COUNT is incremented by 1.

Army Standard: Use of ON SIZE ERROR and ROUNDED should be avoided when possible.

The second form of the ADD verb is used to add two or more things into one or more receiving fields. ROUNDED and ON SIZE ERROR may be specified for each of the receiving fields causing the same treatment as in the first form of the ADD verb.

ADD \{data-name-1 data-name-2 \[...\]} GIVING data-name-m [ROUNDED]
\{numeric-literal-1 numeric-literal-2 \[ON SIZE ERROR any series of COBOL verbs and associated parts except IF-ELSE (period)\]}

EXAMPLE:

ADD IM01-NUMER-OF-DEPENDENTS 1 GIVING WS-FAMILY-SIZE,
ON SIZE ERROR DISPLAY 'FAMILY SIZE TOO BIG'
GO TO 0100-PARA-EXIT.

In this example, IM01-NUMBER-OF-DEPENDENTS and the numeric-literal 1 are added together and the result is placed into WS-FAMILY-SIZE. If WS-FAMILY-SIZE cannot contain the number of digits of the result of the computation, then an ON SIZE ERROR condition exists and the verbs after ON SIZE ERROR until the next period are executed. If ON SIZE ERROR had not been specified and the condition arose during the computation, the result would be truncated and the next statement executed. An advantage of the GIVING option is that none of the original values, except the result, are changed by the ADD.

The third form of the ADD verb adds all of the items under group-item-1 to all of the identically named items under group-item-2. ROUNDED and ON SIZE ERROR operate identically as stated before.

ADD CORRESPONDING group-item-1 TO group-item-2 [ROUNDED]
CORR
\{ON SIZE ERROR any series of COBOL verbs and associated parts except IF-ELSE (period)\}
EXAMPLE:

ADD CORRESPONDING IM01-MASTER-PAY-RECORD TO WS-MASTER-PAY-TOTALS.

This example will add all of the items under IM01-MASTER-PAY-RECORD to the identically named items under WS-MASTER-PAY-TOTALS truncating the results if necessary.

Army Standard: Add corresponding will not be used.

SUBTRACT

SUBTRACT has three forms. They are:

\[
\text{SUBTRACT} \left\{\begin{array}{c}
data-name-1 \\
\text{numeric-literal-1}
\end{array}\right\} \left\{\begin{array}{c}
data-name-2 \\
\text{numeric-literal-2}
\end{array}\right\} \text{ FROM } data-name-m \left[\text{ROUND}ED\right]
\]

\[
\left[\begin{array}{c}
data-name-n \left[\text{ROUND}ED\right] \\
\end{array}\right]...
\]

[ON SIZE ERROR any series of COBOL verbs and associated parts except IF-ELSE (period)]

\[
\text{SUBTRACT} \left\{\begin{array}{c}
data-name-1 \\
\text{numeric-literal-1}
\end{array}\right\} \left\{\begin{array}{c}
data-name-2 \\
\text{numeric-literal-2}
\end{array}\right\} \text{ FROM } data-name-m \text{ numeric-literal-m}
\]

GIVING data-name-n \left[\text{ROUND}ED\right]

[ON SIZE ERROR any series of COBOL verbs and associated parts except IF-ELSE (period)]

\[
\text{SUBTRACT} \left\{\begin{array}{c}
group-item-1 \\
\text{CORRESPONDING}
\end{array}\right\} \text{ FROM } group-item-2 \left[\text{ROUND}ED\right]
\]

[ON SIZE ERROR any series of COBOL verbs and associated parts except IF-ELSE (period)]

In the first form of the SUBTRACT verb, one or more items are added up then the total is subtracted from the item or items following the FROM. ROUNDED and ON SIZE ERROR work identically to the ADD statement.

Army Standard: Use of ON SIZE ERROR and ROUNDED should be avoided where possible.

EXAMPLES:

SUBTRACT 1 FROM WS-TOTAL-PROPLE.
SUBTRACT WS-FICA-TAX WS-SUC-SEC-TAX FROM WS-GROSS-PAY ROUNDED
ON SIZE ERROR MOVE '1' TO WS-ERROR-FLAG
GO TO 9150-PARA-EXIT.
In the second form of the SUBTRACT verb, one or more items are added up then the total is subtracted from the item after the FROM clause and the final result is put into the one or more data-items following the GIVING clause. ROUNDED and ON SIZE ERROR work identically to the ADD statement.

EXAMPLES:

SUBTRACT 1 FROM WS-FAMILY-SIZE GIVING WS-NUMBER-OF-DEPENDENTS.

In both the examples, the item following the FROM is not changed by the computation.

In the third form of the SUBTRACT verb, all of the items under group-item-1 are subtracted from the identically named items under group-item-2. ROUNDED and ON SIZE ERROR work identically to the ADD statement.

EXAMPLE:

SUBTRACT CORR IM³-INVENTORY-ADJUSTMENT-RECORD FROM WS-TOTAL-INVENTORY-DISCREPANCY.

Army Standard: Subtract corresponding will not be used.

MULTIPLY

This verb is used to multiply one by another. It has two forms.

\[
\text{MULTIPLY} \left\{ \begin{array} {c} \text{data-name-1} \quad \text{BY} \quad \text{data-name-2} \\ \text{numeric-literal-1} \end{array} \right\} \quad \text{[ROUNDED]}
\]

[On SIZE ERROR any series of COBOL verbs and associated parts except IF-ELSE (period)]

\[
\text{MULTIPLY} \left\{ \begin{array} {c} \text{data-name-1} \\ \text{numeric-literal-1} \end{array} \right\} \quad \text{BY} \quad \left\{ \begin{array} {c} \text{data-name-2} \\ \text{numeric-literal-2} \end{array} \right\} \quad \text{GIVING} \quad \text{data-name-3} \quad \text{[ROUNDED]}
\]

[On SIZE ERROR any series of COBOL verbs and associated parts except IF-ELSE (period)]

In the first form of the MULTIPLY verb, a number (either a numeric-literal or a data-name) is multiplied by another data-name and the result is put in the data-name after BY. ROUNDED and ON SIZE ERROR work in the same manner as in the ADD statement.

Army Standard: Use of ROUNDED and ON SIZE ERROR should be avoided where possible.

EXAMPLE:

MULTIPLY WS-TAX-RATE BY WS-GROSS-PAY.
This example multiplies the contents of WS-TAX-RATE by the contents of WS-GROSS-PAY and puts the result in WS-GROSS-PAY.

The second form of the MULTIPLY verb is used to multiply two values together putting the result into a data-name. ROUNDED and ON SIZE ERROR work in the same way as in the ADD statement.

EXAMPLE:

MULTIPLY WS-GROSS-PAY BY WS-TAX-RATE GIVING WS-FEDERAL-TAX ROUNDED
ON SIZE ERROR DISPLAY 'FEDERAL TAX OVERFLOW'
GO TO 0210-PARA--EXIT.

DIVIDE

The DIVIDE statement has two forms:

DIVIDE \{data-name-1 \{numeric-literal-1\}\} INTO data-name-2 [ROUNDED]

[ON SIZE ERROR any series of COBOL verbs and associated parts except IF-ELSE (period)]

DIVIDE \{data-name-1 \{numeric-literal-1\}\} INTO \{data-name-2 \{numeric-literal-2\}\} GIVING data-name-3 [ROUNDED]

[REMAINDER data-name-4]

[ON SIZE ERROR any series of COBOL verbs and associated parts except IF-ELSE (period)]

The first form of the DIVIDE verb is used to divide a number or data-name into another data-name. ROUNDED and ON SIZE ERROR work the same way as in the ADD statement. The result is left in the second data-name.

Army Standard: Use of ROUNDED and ON SIZE ERROR should be avoided where possible.

EXAMPLES:

DIVIDE 10 INTO WS-PAGE-TOTAL.
DIVIDE WS-TAX INTO WS-PAY ROUNDED.

In both examples, the result is left in the second variable (after the INTO).

The second form of the DIVIDE verb divides a number or data-name into or by another number or data-name giving a third number or data-name. ROUNDED and ON SIZE ERROR work in the same manner as in the ADD statement.
EXAMPLES:

DIVIDE WS-TAX INTO WS-PAY GIVING WS-TAX-RATE.
DIVIDE A BY B GIVING C ROUNDED.

In both examples, the result is placed in the variable after the GIVING leaving
the first two variables unchanged.

**COMPUTE**

**COMPUTE** identifier-1 [ROUNDED] = \{arithmetic-expression\}
\{identifier-2
\{literal-1

[ON SIZE ERROR any series of COBOL verbs and associated parts except IF-ELSE (period)]

The **COMPUTE** statement gives COBOL powers similar to that of FORTRAN: a complete
arithmetic expression may be evaluated and the result placed in identifier-1.

There are five (5) arithmetic operators that may be used in arithmetic
expressions. Each is represented by a specific character or character
combination that must be preceded by a space and followed by a space, except
that a sign (- or +) must not be preceded by a space when it follows a left
parenthesis.

<table>
<thead>
<tr>
<th>ARITHMETIC OPERATOR</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>**</td>
<td>exponentiation</td>
</tr>
</tbody>
</table>

Parenthesis may be used in arithmetic expressions to specify the order in
which elements are to be evaluated. Expressions within parenthesis are
evaluated first. When parenthesis are not used or parenthetical expressions
are at the same level of inclusiveness, the following heirarchical order is
implied:

1. - or + (indicating negative or positive number)
2. **
3. * and /
4. + and -

When the order of consecutive operations on the same heirarchical level is
not completely specified by parentheses, the order of operation is from left
to right.

**ON SIZE ERROR** and **ROUNDED** operate identically to the **ADD** statement.
Army Standard: Use of the ON SIZE ERROR and ROUNDED should be avoided where possible.

EXAMPLE: COMPUTE WS-RESULTS ROUNDED = (A + B - C * D) - (-E / F + (G * H ** I))

ORDER OF OPERATION:

H is raised to the Ith power
G is multiplied by the result of (H**I)
E is negated
F is divided into E
The result of -E/F is added to G*H**I
C is multiplied by D
A is added to B.
C*D is subtracted from A+B
The result of -E/F+(G*H**I) is subtracted from the result of A+B-C*D
This result is rounded up and the result if placed into WS-RESULTS.
A B C D E F G H and I remain unchanged.

NOTE: Army naming convention is dropped for this example for illustrative clarity.

INPUT-OUTPUT
READ

The form of the READ statement is:

READ file-name

AT END any series of COBOL verbs and associated parts except the IF or IF-ELSE ended by a period.

This verb reads any sequential file (cards, disk, or tape file) and places the information that was read into the record description under the file-name in the FD. Each time this verb is executed, one record is read from the file. After the last record has been READ, the next time the READ statement is executed, the AT END series of verbs is executed. Care should be exercised to insure that once the AT END condition arises the file is never READ again except when another file immediately follows the one just processed. If a record was actually READ, the period (after the AT END series of verbs) is executed.

EXAMPLE:

READ CARD-FILE
AT END MOVE 'OFF' TO WS-READ-END-OF-FILE-SWITCH
GO TO 1450-PARA-EXIT.
MOVE IM03-NAME TO WS-NAME.
In the example, a card will be READ each time the READ statement is executed. After the last card is READ and the statement is executed once more, 'OFF' will be MOVED to WS-READ-END-OF-FILE-SWITCH and control will go to the paragraph exit. If the AT END condition did not occur when the READ was executed, the MOVE statement following the period after the GO TO 1450-PARA-EXIT would be executed. If the period after the GO TO 1450-PARA-EXIT was accidentally omitted, the MOVE statement would be considered part of the AT END process and the statement after the MOVE would not be executed following a good READ. Likewise, if the period was placed after the MOVE 'OFF' statement, then the GO TO 1450-PARA-EXIT would be executed for each good READ. Clearly, placement of period after AT END is important to program execution.

Army Standard: (1) Only one READ per file is normally required. This should be in a one statement paragraph that is performed from the needed places in the program. (2) Never execute a READ for a file which has encountered end-of-file unless it has been closed and reopened as input.

WRITE

A form of the WRITE verb is:

WRITE file-record [FROM record-name] BEFORE ADVANCING data-name LINES. AFTER integer mnemonic-name

This form writes to the printer or card-punch. Each time the WRITE statement is executed, the information in the record under the file description is written to the device specified (either a card punch or line printer). If the FROM option is included, the information in the record-name is copied to the file-record under the file description and then written. If the punch is being written to, then the BEFORE option should be chosen if the programmer wants to determine which punch pocket the cards should be placed in after it is physically punched. A mnemonic name associated with the punch pocket in the SPECIAL-NAMES paragraph is used to select the actual pocket. If a printer is being written to, either BEFORE or AFTER may be used and the corresponding writing will take place. If BEFORE is specified, then the line being written will be written before the spacing. If AFTER is specified, then the line will be written after the spacing. Spacing is controlled by the ADVANCING clause. If a data-name is used, then it must be a non-negative elementary numeric item (normally a 77 entry) whose value lies between 0 and 99. If an integer is chosen, the same constraints on the values apply as with the data-name. If a mnemonic-name is chosen, then the mnemonic-name used must be defined in the SPECIAL-NAMES paragraph and associated with one of the printer carriage control channels.
EXAMPLES:

1. Write to the card punch:

SPECIAL-NAMES.
  S01 IS TO-POCKET-1.

  WRITE OP01-CARD-PUNCH BEFORE ADVANCING TO-POCKET-1.

or

  WRITE OP01-CARD-PUNCH.

2. Write to the printer.

SPECIAL-NAMES.
  C01 IS CHAN1.

  WRITE OR01-PRINTER-RCD AFTER ADVANCING CHAN1.

or

  WRITE OR01-PRINTER-RCD FROM WS-DETAIL-LINE AFTER ADVANCING 2 LINES.

or

  77 WS-CARR-CONTROL          PIC 99
      VALUE 4.

  WRITE OR01-PRINTER-RCD BEFORE ADVANCING WS-CARR-CONTROL LINES.

In the first example, TO-POCKET-1 is associated with pocket one (S01) in the card punch. The write statement forces the punched card to be placed into pocket 1 through use of the BEFORE ADVANCING TO-POCKET-1 clause. In the second case of writing to the card punch, the programmer simply allows the system to choose the pocket for the card to be placed in. Pocket selection is normally done when two or more types of cards are being punched and are to be kept separate. When the ADVANCING option is used, an extra character at the left of the record must be reserved for the system to use; e.g., to punch an 80 character record, 81 characters must be defined with the first character being a FILLER. When the BEFORE ADVANCING option is not used, then only 80 characters must be defined.

In the second example, CHAN1 is associated with printer carriage channel 1. Printer carriage channel 1 is almost always lined up with line six of the printer page. When, as in the first WRITE statement, the line is written AFTER ADVANCING CHAN1, the paper in the printer is advanced until printer carriage channel 1 is sensed then the line is written. This allows the
programer to begin headings at the top of a new page. In the second case, the information in WS-DETAIL-LINE is moved to OR01-PRINTER-RCD then OR01-PRINTER-RCD is written after double spacing past the last line. In the third case, a data item is used to determine the spacing. The value that the printer is advanced BEFORE or AFTER writing the line depending on the option chosen. As in the card punch, whenever the ADVANCING option is used, an extra character on the left of the line must be reserved for system use. The IBM 1403 printer has 132 printable positions. When ADVANCING is used on this device, 133 characters must be written. The first character of the 133 is used by the system and the remaining 132 are printed.

Another form of the WRITE statement is:

```
WRITE file-record [FROM record-name].
```

This form is used to write to tape files. There is no BEFORE or AFTER option. The FROM clause is optional.

**EXAMPLES:**

```
WRITE OT01-TAPE-FILE-RCD.
```

```
WRITE OT01-TAPE-FILE-RCD FROM WS-ADD-RECORD.
```

In the first example, all of the information to be written to the tape file had been moved into OT01-TAPE-FILE-RCD (the Ø1 record description under the FD for the tape file) prior to the WRITE. In the second case, the information had been moved from WS-ADD-RECORD to OT01-TAPE-FILE RCD prior to WRITE. WS-ADD-RECORD would normally be defined in the WORKING-STORAGE SECTION somewhere.

The last form of the WRITE statement is:

```
WRITE file-record [FROM record-name] INVALID KEY a series of COBOL verbs and associated parts except IF-ELSE (period).
```

This form is used to write to disk files. File-record is the Ø1 under the FD for the disk file. The FROM record-name is optional and is used identically as the other two examples. INVALID KEY is used to tell the programer when the disk file is full and no more records can be written to it. Disks files have a definite amount of space to hold the information they are given. When that space is all used, then the program has to be informed of it. This is done via the INVALID KEY clause. When the file is full and one more record is written, the series of COBOL verbs and associated parts after the INVALID KEY is executed. An IF verb cannot be one of the verbs used in the series.

**Army Standard:** (1) Only one WRITE for each record of a different size should be in a program. Each WRITE should be a one statement paragraph that is PERFORMed from the appropriate places.
(2) For files with several types of records, it is best to define them in WORKING-STORAGE and move them to the record area just before the WRITE.

DISPLAY

DISPLAY is a useful verb for debugging the program. It should be used to print intermediate results and values during the run of a program. It relieves the programmer of defining a printer file during the debugging process and then having to pull all of the cards defining that file out of the program after the debugging phase is over. Its form is:

DISPLAY a series of literals (things enclosed in quotes) and data-names.

If the series is over 120 characters in length, only 120 characters will be printed on a line and the rest will be printed on the next line.

EXAMPLE:

DISPLAY 'NEW MASTER RECORD IS: ' OM01-NEW-MASTER-RECORD.

This example will print the literal 'NEW MASTER RECORD IS: ' followed by the characters in the OM01-NEW-MASTER-RECORD all on one line. Because the literal is 22 characters long, the number of characters in OM01-NEW-MASTER-RECORD that can be printed on the printer is 98. If OM01-NEW-MASTER-RECORD is longer than 98 characters, then only the first 98 characters are printed on this line and the rest are printed on the next line. Each time a DISPLAY is executed, the contents of the series of literals and variables are begun on a new line single spaced after the last line. There is no way to go to the top of a page with a DISPLAY statement. Double spacing can be achieved by simply displaying a blank before or after each real DISPLAY.

EXAMPLE:

DISPLAY '.
DISPLAY 'TOTAL NUMBER OF PEOPLE IS: ' WS-PEOPLE-TOTAL.

This example prints a blank line then a line giving the total people message.

Army Standard: Display should only be used as a debugging tool except in communications with the operator.

EXHIBIT (IBM Extension)

The EXHIBIT verb is another debugging tool. Its form is:

EXHIBIT NAMED series of data-names and literals.
This verb formats each of the data-names into a DATA-NAME=value, form. These are spread out across the page. If the list is too long to fit on one line, then it is carried over to the next line. When the EXHIBIT NAMED form is used, then every data-name in the list is listed each time the EXHIBIT is executed. When the EXHIBIT CHANGED form is used, all of them are reported the first time the verb is executed. The next time the verb is executed, only those data-names that have changed since the last execution are listed.

Army Standard: EXHIBIT should only be used as a debugging tool.

EXAMPLE:

EXHIBIT NAMED WS-DIVISION-HOLD WS-LINE-COUNT.

EXHIBIT CHANGED WS-DIVISION-HOLD WS-LINE-COUNT.

In the first example, WS-DIVISION-HOLD and WS-LINE-COUNT will be listed each time the statement is executed. In the second example, both WS-DIVISION-HOLD and WS-LINE-COUNT are printed the first time the EXHIBIT is executed. From then on whenever the statement is executed, only the data item that has changed (one or both) will be printed. If neither one has changed, then nothing is printed.

OPEN

The OPEN verb is used to prepare the files for input or output. Execution of this verb must precede the execution of any READ or WRITE statement. Its form is:

OPEN INPUT one or more file-names to be opened as either INPUT or OUTPUT

OUTPUT

INPUT one or more file-names to be opened as either INPUT or OUTPUT...

OUTPUT

EXAMPLES:

OPEN INPUT PAY-MASTER-FILE.
OPEN INPUT CARD-READER DISK-FILE
OUTPUT PRINTER NEW-DISK-FILE.
OPEN OUTPUT CARD-PUNCH PRINT-FILE.

The first example opens the PAY-MASTER-FILE as an input file and prepares that file for processing. The second example opens the CARD-READER and DISK-FILE as input files then opens the PRINTER and NEW-DISK-FILE as output files. The third example opens the CARD-PUNCH and PRINT-FILE as output files. Care should be exercised not to open a file more than once unless a CLOSE (described next) for that file has been executed in the interim. It is completely legal and necessary sometimes to OPEN a file as output then when processing is complete on that file, to close it and open it as input. All of the file names must be described in an FD in the FILE SECTION.
Army Standard: As many files as possible should be opened by one OPEN statement.

CLOSE

The CLOSE verb is used to disassociate a file with a program. Before the program is finished execution, all of the files currently open should be closed. If it is necessary to access a file as both input and output at different times, a CLOSE statement must be executed between the two types of access. The form of the CLOSE verb is:

CLOSE one or more file-names

EXAMPLES:

CLOSE INPUT-MASTER PRINTER.
CLOSE PRINTER.
CLOSE CARD-READER
    DISK-FILE.

All of the examples simply CLOSE the list of files after the CLOSE verb.

Army Standard: As many files as possible should be closed by one CLOSE statement.

LOGIC FLOW CONTROL VERBS

Paragraph names, while not verbs, are used by the logic control verbs in branching. They are margin entries and are used to identify logically significant points in the program. They follow the same naming convention as data names; however, they may be completely numeric also. This is not recommended. They are ended with a period.

EXAMPLES:

0170-READ-AND-PRINT.
    
    
0170-RAP-EXIT.

IF is used to make decisions during the execution of the program. Its form is:

IF condition any series of legal COBOL verbs and associated parts.
EXAMPLE:

IF IM01-SOC-SEC-NUM EQUAL TO IT01-SOC-SEC-NUM
   PERFORM 0100-UPDATE-Routine THRU 0100-UPDATE-EXIT
   PERFORM 0200-READ-NEW-MASTER THRU 0200-READ-EXIT
   PERFORM 0300-READ-UPDATE THRU 0300-READ-EXIT.

In the above example, if IM01-SOC-SEC-NUM is equal to IT01-SOC-SEC-NUM, then the 0100-UPDATE-Routine is PERFORMed, the 0200-READ-NEW-MASTER routine is PERFORMed, then the 0300-READ-UPDATE routine is PERFORMed. The entire series is executed only if the condition is true. If the condition is not true, then the sentence after the period ending the IF is executed. It is important to note that the first period after the IF ends that IF statement.

EXAMPLE:

IF A IS LESS THAN 0
   ADD 1 TO X
   ADD 2 TO Y.
   ADD 3 TO Z.

IF A IS LESS THAN 0
   ADD 1 TO X.
   ADD 2 TO Y
   ADD 3 TO Z.

NOTE: Army naming convention is not used for this example for illustrative purposes.

In the first case, if A is less than zero, one is added to X and two is added to Y then three is added to Z. If A is not less than zero, the adds to X and Y are skipped, then three is added to Z. The period ends the control of the IF. When the condition after the IF is true, the statements under control of the IF (e.g., the statements between the condition and the period) are executed. When the condition is not true, the statements under control of the IF are skipped and execution begins with the first statement after the period.

In the second case, if A is less than zero, one is added to X, the two is added to Y, and three is added to Z. If A is not less than zero, the add to X is skipped, then two is added to Y and three is added to Z.

Periods are very important in the logical construction of an IF statement. The omission or addition of a period can radically effect the execution of a program.

EXAMPLE:

IF A IS LESS THAN B
   ADD 1 TO A-COUNT
   GO TO A-EXIT.
   ADD 1 to B-COUNT.

or
IF A IS LESS THAN B
   ADD 1 TO A-COUNT.
   GO TO A-EXIT.
ADD 1 TO B-COUNT.

NOTE: Army naming conventions are not used in this example for illustrative purposes.

In the first case, the branch to A-EXIT is only taken if A is less than B. If A is not less than B, then the statement after the period ending the IF is executed (ADD 1 TO B-COUNT). In the second case, if A is less than B or not less than B, the branch to A-EXIT is executed since the period ending the IF statement is after the ADD 1 TO A-COUNT. B-COUNT will never be incremented since execution cannot reach that point.

Another form of the IF statement is:

IF condition any series of COBOL verbs with associated parts (no periods)
ELSE any series of COBOL verbs with associated parts.

EXAMPLE:

IF IM01-DIVISION NOT = WS-DIVISION-SAVE
   OR WS-LINE-COUNT GREATER THAN 45
   PERFORM 0150-HEADING-ROUTINE THRU 0150-HEADING-EXIT
   MOVE 1 TO WS-LINE-COUNT
ELSE
   ADD 1 TO WS-LINE-COUNT.
   MOVE IM01-NAME TO WS-NAME.

In the above example, the first condition was made up of two parts (a compound condition) consisting of IM01-DIVISION NOT = WS-DIVISION-SAVE and WS-LINE-COUNT GREATER THAN 45. The OR between the two tests means that if either (or both) of the tests is true, then everything until the ELSE is to be executed in order. If both of the tests are false, then everything after the ELSE will be executed instead. Then control is given to the sentence after the period (a MOVE statement).

Another connector between tests is the word AND. AND indicates that both conditions have to be true for the statements under the IF to be executed. If they are not true (either or both) then execution is given to the statements after the ELSE if present. If there is no ELSE associated with the IF, then execution goes to the first statement after the period ending the IF.
EXAMPLE:

IF IM01-DIVISION = WS-DIVISION-SAVE
   AND IM01-SOC-SEC-NUM EQUAL TO IT101-SOC-SEC-NUM
   PERFORM 0101-UPDATE-ROUTINE THRU 0101-UPDATE-EXIT
ELSE IF IM01-DIVISION = WS-DIVISION-SAVE
   AND IM01-SOC-SEC-NUM GREATER THAN IT201-SOC-SEC-NUM
   PERFORM 0102-ADD-ROUTINE THRU 0102-ADD-EXIT.
MOVE IM01-NAME TO WS-NAME.

In the above example, the first IF has a compound condition consisting of whether IM01-DIVISION was EQUAL TO WS-DIVISION-SAVE AND also whether IM01-SOC-SEC-NUM was EQUAL TO IT101-SOC-SEC-NUM. If both tests were true, then 0101-UPDATE-ROUTINE is PERFORMed. If either or both tests were false, then the statements after the ELSE are executed. The second IF also is made up of two tests: whether IM01-DIVISION is EQUAL TO WS-DIVISION-SAVE AND whether IM01-SOC-SEC-NUM is GREATER THAN IT201-SOC-SEC-NUM. If both tests were true, then 0102-ADD-ROUTINE is PERFORMed. If either or both of the tests were false, then the MOVE statement after the period ending the entire IF is executed.

Notice that everything from the first IF to the final period is all one sentence (one logical construction). If any period had been in the middle of the sentence, then the IF will have ended there. Periods are very important things in IF statements. Notice also that one can (and many times should) put another IF as one of the COBOL verbs whose execution depends on the trueness or falseness of the original IF. This is called nesting IF's. In the above example, the test for equality between IM01-DIVISION and WS-DIVISION-SAVE is made twice. An example of how to avoid this and still have the same logical construction is given below.

EXAMPLE:

IF IM01-DIVISION = WS-DIVISION-SAVE
   IF IM01-SOC-SEC-NUM = IT101-SOC-SEC-NUM
      PERFORM 0101-UPDATE-ROUTINE THRU 0101-UPDATE-EXIT
   ELSE
      IF IM01-SOC-SEC-NUM GREATER THAN IT201-SOC-SEC-NUM
         PERFORM 0102-ADD-ROUTINE THRU 0102-ADD-EXIT.
   MOVE IM01-NAME TO WS-NAME.

In the above example, the first IF has only one test: whether IM01-DIVISION is EQUAL TO WS-DIVISION-SAVE. Everything else in the sentence is dependent on the trueness or falseness of that test. If it was false, then control is given to the MOVE statement after the period. If it was true, then the second IF is executed. If the second condition was true, then 0101-UPDATE-ROUTINE is PERFORMed. After 0101-UPDATE-ROUTINE is finished, control is given to the MOVE statement after the period. If the second condition is false, then the third IF is executed. If that condition is true, then 0102-ADD-ROUTINE is PERFORMed. After 0102-ADD-ROUTINE is finished, control is given to the MOVE statement after the period. Notice that the first and third IF's have no ELSE associated with them. The rule for matching IF's with ELSE's is that an ELSE is associated with the first possible IF above the ELSE.
This allows the programer to build very complex logical constructions with compound conditions and nested IF's. Tremendous care should be exercised when designing these constructions.

If necessary, parentheses should be used in compound conditions to assure evaluation in the proper order. If OR and AND are used in the same compound, then the AND test is evaluated first.

EXAMPLE:

IF A IS NOT GREATER THAN B
   OR A + B IS EQUAL TO C
   AND D IS POSITIVE

will be evaluated as though it were parenthesized as follows:

IF (A IS NOT GREATER THAN B)
   OR (((A + B) IS EQUAL TO C)
   AND (D IS POSITIVE))

NOTE: Army naming conventions are not used for these examples for illustrative purposes.

The use of parentheses never will hurt a program's efficiency and can only add to the clarity of what the programer is trying to do. Also when going from one computer to another manufacturers computer, the evaluation rules could conceivably differ. With parentheses, there is no doubt for either the compiler or another programer having to make changes to the program in the absence of the original programer.

If at some point inside the nested IF's the programer would like to simply exit from the entire construction and go to the sentence after the period, the COBOL reserved phrase NEXT SENTENCE may be used.

EXAMPLE:

IF condition-1
   IF condition-2
      IF condition-3
         NEXT SENTENCE
      ELSE
         IF condition-4
            ..
      ELSE NEXT SENTENCE
   ELSE
      IF condition-b
         ..
   ELSE NEXT SENTENCE

DRAFT ST 18-150
ELSE

NEXT SENTENCE.
In the above example, if conditions 1, 2, and 3 are true, then control is simply given to the sentence after the period (the periods inside the nested IF's in the example only indicate a series of COBOL verbs and associated parts). If conditions 1 and 2 are true and conditions 3 and 4 are false, then again control is simply passed to the sentence after the period. If condition 1 is false, then the associated ELSE means to execute the sentence after the period. The last ELSE NEXT SENTENCE (period) is superfluous and was included to show the point. If the ELSE was not there and condition 1 was false, control would be given to the sentence after the period anyway. Some programers consider it good practice to match all IF verbs with an ELSE even if the ELSE is superfluous as in this case. If NEXT SENTENCE is put immediately after an ELSE or the conditions following an IF, no other verb except ELSE may follow it. Putting NEXT SENTENCE after a series of verbs and associated parts simply tells the compiler to do what it would have done anyway.

EXAMPLE:

IF condition-1
   
   NEXT SENTENCE
ELSE
   
In the above example, NEXT SENTENCE is unnecessary. Control will go to the next sentence anyway.

88 level items can be used in conjunction with an IF.

EXAMPLE:

03 WS-PAY PIC 9(6)V99.
   VALUE .00 THRU 100.00.
   
   IF SMALL
      PERFORM 0505-SMALL-PAY-PROCESSING THRU 0505-SMALL-EXIT
ELSE
   
   is equivalent to:

IF (WS-PAY EQUAL TO .00 OR WS-PAY GREATER THAN .00)
   AND (WS-PAY LESS THAN 100.00 OR WS-PAY EQUAL TO 100.00)
   PERFORM 0505-SMALL-PAY-PROCESSING THRU 0505-SMALL-EXIT
ELSE
   
   

220

DRAFT ST 18-150
Clearly it is more convenient to use the 88 level item and there is the added benefit of being able to change the ranges of things being tested for by simply changing one card. 88's also increase the readability of the program and are self documenting with properly chosen names.

Army Standard:
(1) Use of the nested IF other than by the original programer is prohibited.
(2) Compound conditions requiring use of both AND and OR are prohibited.
(3) NOT will not be used in compound conditions.
(4) Avoid:
   (a) Using the NUMERIC and ALPHABETIC test unnecessarily.
   (b) Comparing numeric items with different number of decimal places. Move one of the items to a holding field that has the same PICTURE instead.
   (c) Comparing non-numeric items with different size PICTURES.
   (d) Comparing group items that have unused areas in them.
   (e) Using the phrase ELSE NEXT SENTENCE unless it is in a nested IF.
(5) Do not specify a GO TO in and IF that goes to some code then returns the statement after the IF. Use a PERFORM instead or incorporate the code into the IF itself.
(6) NEXT SENTENCE can often be avoided by reversing the condition in the IF (using a NOT).
(7) Don't test the same data item for the same value once you have determined what the value is.
(8) Try to test for ranges of values instead of specific values, when appropriate.
(9) Don't use the ALPHABETIC test. Use IF...NOT NUMERIC instead.

EXAMPLES:

<table>
<thead>
<tr>
<th>More efficient</th>
<th>Less efficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1 IF A EQUALS B</td>
<td>IF A EQUALS B</td>
</tr>
<tr>
<td>MOVE X TO Y</td>
<td>GO TO C.</td>
</tr>
<tr>
<td>PERFORM Z THRU Z-EXIT.</td>
<td>D. MOVE ...</td>
</tr>
<tr>
<td>MOVE ...</td>
<td></td>
</tr>
<tr>
<td>ELSE</td>
<td></td>
</tr>
<tr>
<td>IF A NOT EQUAL TO B</td>
<td>IF A EQUALS B</td>
</tr>
<tr>
<td>MOVE C TO D.</td>
<td>NEXT SENTENCE</td>
</tr>
<tr>
<td>ELSE</td>
<td>ELSE MOVE C TO D.</td>
</tr>
<tr>
<td>IF A EQUALS 1</td>
<td>IF A EQUALS 1</td>
</tr>
<tr>
<td>MOVE X TO B</td>
<td>MOVE X TO B.</td>
</tr>
<tr>
<td>ELSE</td>
<td>IF A EQUALS 7</td>
</tr>
<tr>
<td>IF A EQUALS 7</td>
<td>MOVE Y TO B.</td>
</tr>
<tr>
<td>MOVE Y TO B</td>
<td>IF A EQUALS 9</td>
</tr>
<tr>
<td>ELSE</td>
<td>MOVE Z TO B.</td>
</tr>
<tr>
<td>IF A EQUALS 9</td>
<td></td>
</tr>
<tr>
<td>MOVE Z TO B.</td>
<td></td>
</tr>
</tbody>
</table>
PERFORM

The PERFORM statement has a number of forms each having its own application. They will be discussed in detail one at a time. The basic concept of a PERFORM is common for all of them. While writing a program, that program can always be broken up into several discrete parts which function independently of each other. The main control section of the program simply gives control to the appropriate part at the appropriate time through a series of PERFORM statements. When the part that has been given control is finished, control is given to next COBOL verb after the PERFORM statement.

EXAMPLE:

PERFORM 0100-ADD-RUTINE THRU 0100-ADD-EXIT.
CLOSE ADD-FILE.

0100-ADD-RTN.
MOVE IT01-ADD-RCD TO OM01-NEW-MASTER-RECORD.
WRITE OM01-NEW-MASTER-RECORD.
0100-ADD-EXIT. EXIT.
0200-SUBTRACT-RTN.

In the above example, when control is passed to the PERFORM 0100-ADD-RTN THRU 0100-ADD-EXIT statement, the PERFORM statement passes control to 0100-ADD-RTN. IT01-ADD-RCD is moved to OM01-NEW-MASTER-RECORD and then OM01-NEW-MASTER-RECORD is written. As soon as the 0100-ADD-EXIT paragraph is executed, the COBOL verb EXIT tells the machine to transfer control back to the PERFORM which passed control to the 0100-ADD-RTN. The PERFORM then passes control to the next COBOL verb after the PERFORM (CLOSE ADD-FILE). The PERFORM is really a 'go do this then come back when your finished' verb. A simpler version, but less preferable, is:
EXAMPLE:

PERFORM 0100-ADD-RTN.
CLOSE ADD-FILE

0100-ADD-RTN.

MOVE ADD-RECORD TO NEW-MASTER-RECORD.
WRITE NEW-MASTER-RECORD.

Army Standard: An EXIT paragraph should always be the last paragraph in a PERFORM statement.

In the above example, exactly the same thing happens as the last example. While before the EXIT was explicitly defined in the PERFORM statement and written at the end of the paragraph that was being PERFORMED, this time the EXIT is implied to be at the end of the ADD-RTN paragraph just before the next paragraph name (0200-SUBTRACT-RTN.) and after the last period at the end of the last sentence in ADD-RTN (WRITE NEW-MASTER-RECORD.). It is good programing procedure to always use an explicit EXIT and PERFORM a paragraph through that EXIT. This makes the program more readable and makes the job of a maintenance programmer fixing the program at a later date much easier.

Another form of the PERFORM verb is:

PERFORM paragraph-name [THRU paragraph-exit]
UNTIL condition.

This form will PERFORM the paragraph as many times as necessary to make the condition true. This form can make an indefinite loop if the condition is never true. An example of this would be if the condition was A = B AND A NOT = B. This can never be true so the paragraph will be executed forever or until the operator gets tired of looking at the program executing. Clearly, this is not a desirable thing to do. Care should be used to insure that the condition will be made true at some point in the processing. If the condition is true when the PERFORM is executed, then the paragraph is never executed at all and control passes to the next verb. The condition may be an 88 level item. See the IF statement discussion for what it means.

Army Standard: (1) Always use PERFORM...THRU.
(2) Always execute the EXIT paragraph to return to the PERFORM.

EXAMPLE:

01 WS-EOF-TABS.
03 MASTER-EOF-SWITCH 88 MASTER-DONE
   PIC X VALUE '0'.
   VALUE '1'.

PERFORM 0010-PARA-1 THRU 0010-PARA-EXIT
UNTIL MASTER-DONE.
This example will PERFORM 0010-PARA-1 THRU 0010-PARA-EXIT until '1' is moved to WS-MASTER-EOF-SWITCH. When this happens, the 88 under WS-MASTER-EOF-SWITCH becomes true and the UNTIL is satisfied.

PERFORM paragraph-name [THRU paragraph-exit] n TIMES.

This form of the PERFORM verb will PERFORM the paragraph a fixed number of times where n is an integer that specifies how many times the paragraph is to be executed. If n is a variable, then it must be an integer.

EXAMPLE:

77 PERFORM-COUNT PIC S9(4) USAGE IS COMP VALUE IS +20 SYNC.

PERFORM 5001-PARA-1 THRU 5001-PARA-EXIT
PERFORM-COUNT TIMES.

or

PERFORM 5001-PARA-1 THRU 5001-PARA-EXIT
20 TIMES.

Both of the examples do the same thing. They will PERFORM 5001-PARA-1 through 5001-PARA-EXIT 20 times.

The last form of the PERFORM verb is:

PERFORM paragraph-name [THRU paragraph-exit] VARYING name-1 FROM value-1 BY value-2 UNTIL condition.

This form will initialize name-1 (index-name-1 or numeric-data-item-1) to value-1 (integer-1, index-name-2, or numeric-data-item-2) and test the condition (a simple or compound condition). If the condition is false, the paragraph is PERFORMed. When the paragraph is done the first time, name-1 is incremented by value-2 (integer-2, index-name-3, or numeric-data-item-3) and the condition is tested again. This process of PERFORMing the paragraph, incrementing name-1 and testing the condition continues until the condition is true. If the condition is true when the PERFORM is executed the first time, then control immediately passes to the next verb and the paragraph is never performed.

EXAMPLE:

PERFORM 0100-PARA-1 THRU 0100-PARA-EXIT
VARYING TABLE-INDEX FROM 1 BY 1
UNTIL TABLE-INDEX IS GREATER THAN 25.
This example initializes TABLE-INDEX to 1, then tests whether TABLE-INDEX is greater than 25. If it is not so, $0100$-PARA-1 is executed. When control finally reaches $0100$-PARA-EXIT, TABLE-INDEX is incremented by 1 (giving 2), and the condition is again evaluated. TABLE-INDEX is still not greater than 25, so $0100$-PARA-1 again gets control; however, TABLE-INDEX is now equal to 2. This type of PERFORM is very useful in searching tables for an entry. More of how to use this form will be discussed in the section on tables.

EXIT

EXIT is a verb that provides a way to terminate a paragraph being PERFORMed and then return to the PERFORM statement for further processing if necessary. It can be the only thing after the paragraph-name other than comments ("*" in column 7) until the next paragraph-name.

GO TO

The GO TO is the most abused single verb in COBOL. When an error is found in a program, the average programer immediately begins to think in terms of GOing somewhere else. This leads to many unpredictable results initially and usually has some hidden implications that may not show up until the next programer has to repair the program again. A study was done to try to determine the cause of COBOL logical errors. The result of this study was to find the GO TO guilty of the vast majority. In this context, this text will present the way a GO Tu may be used and then place several large constraints on how to use it. The form of the GO TO is:

GO TO paragraph-name.

EXAMPLE:

GO TO 0100-PARA-1.

In this example, the sentence immediately following 0100-PARA-1 will be executed. Indiscriminate use of this type of statement can lead to 'spaghetti' logic that is hard to debug and repair at a later date. A much more restrictive use is:

EXAMPLE:

GO TO 0100-PARA-EXIT.

This is the only type of GO TO allowed in 'structured' programing. In a given paragraph, the programer can only go to the EXIT of that paragraph.
EXAMPLE:

Ø525-PARA-1.
IF LINE-COUNT GREATER THAN 20
    GO TO Ø525-PARA-EXIT.
Ø525-PARA-EXIT. EXIT.

This type of limitation requires that each paragraph be entered via a PERFORM statement. Most paragraphs won't need a GO TO in them at all. However, at some future date, another programmer may find it necessary to put one in. To keep the program compatible, the initial programmer should always have an EXIT for each paragraph and PERFORM THRU that exit. If the PROGRAM was filled with a mixture of PERFORMs and PERFORM THRUss and suddenly another programmer needed to change one of the PERFORMs to a PERFORM THRU, all of the PERFORMs for that paragraph would have to be changed, leading to a lot of work and possible source of new problems. If all of the paragraphs have EXIs, then the programmer simply GO TOs the EXIT and makes no other changes to the program. This cuts down on maintenance time, a major cause of cost overrun on a project and also makes the original programmer spend more time on the design of the program leading to more trouble free code. Some programmers complain that COBOL is already too verbose and wordy and all of these restrictions make his job harder and longer. That may be true, but the real cost of a program is not in the writing of it; it is in the maintenance of it. By forcing the programmer to conform to this type of design forces him to use routine (break the program up into pieces that do discrete, separate things). This makes the problem clearer and the logic simpler.

STOP

This verb is used to either temporarily halt a program or to terminate it altogether. The form of the STOP statement is:

STOP RUN quoted-literal

EXAMPLES:

STOP RUN.
STOP 'MOUNT 2402-TEST-DATA-TAPE'

The first example terminates the program. The second example halts the program and types the literal on the console typewriter. The operator will then restart the program after appropriate action has been taken on the message typed.

Army Standard: (1) There should be only one STOP RUN in the program. This should be a single statement paragraph.
(2) All files must be closed before the STOP RUN is executed.
The ON verb allows the programer to specify when a series of COBOL verbs are to be executed. Its form is:

ON integer-1  [AND EVERY integer-2] [UNTIL integer-3]
any series of COBOL verbs and associated parts except IF-ELSE
[ELSE]
[OTHERWISE]
NEX SENTENCE
any series of COBOL verbs including IF-ELSE

This verb sets up a count-condition. If only integer-1 has been specified, then the count-condition will only be satisfied once. The count is incremented by one each time the ON statement is executed. If the count-condition is not satisfied, the series of COBOL verbs after the ELSE is executed. If no ELSE is specified and the count-condition is not satisfied, execution goes to the next sentence after the period terminating the ON statement.

Army Standard: ON should only be used during debugging.

EXAMPLE:

ON 1 MOVE ZEROS TO WS-LINE-COUNT
PERFORM 5626-HEADING-ROUTINE THRU 5626-HEADING-EXIT.

ON 10 AND EVERY 10
PERFORM 1000-SUB-TOTAL-ROUTINE THRU 1000-SUB-EXIT.

ON 1 UNTIL 1000
READY TRACE
ELSE RESET TRACE.

ON 25 AND EVERY 25 UNTIL 975
NEXT SENTENCE
ELSE PERFORM 0500-PROCESS-ROUTINE THRU 0500-PROCESS-EXIT
PERFORM 0600-COUNT-ROUTINE THRU 0600-COUNT-EXIT.

The first example executed the MOVE and PERFORM the first time the statement is executed. The second example will execute the PERFORM every ten times the statement is executed. The third example initiates a TRACE (described next) the first 1000 times the statement is executed. After the 1000th time, the ELSE path is taken and the TRACE is terminated. The fourth example simply goes to the period after the statement on the 25th time and each subsequent 25th time until the ON statement has been executed 975 times. If the particular time the ON is being executed is not one of the 25th times or is past the 975 limit, then the two PERFORMs under the ELSE are executed. This statement works very similarly to the IF-ELSE statement where the condition is described by the ON.
DEBUGGING

Tracing is a powerful tool for the programmer. The format of the TRACE statement is:

\[ \text{READY TRACE (IBM EXTENSION)} \]

\[ \text{RESET} \]

EXAMPLES:

READY TRACE.
RESET TRACE.

When the first example is executed, the COBOL listing number (number to the extreme left on the listing) of the paragraphs will be listed on the printer as they are executed. Any lines the program writes on the printer will be interspersed at the appropriate places. This allows the programmer to actually see the order that the paragraphs in his program were executed and to hopefully find out why the logic is going awry. The second example turns off the trace. If the trace is off when it is turned off again, nothing happens. Likewise if the trace is on when it is turned on again, nothing happens.

The TRACE verb can generate a large amount of paper and should be used with discretion. Intermediate values can be printed using the DISPLAY verb or the EXHIBIT verb during the run while using the trace and a clearer picture of what is going on is made available. A powerful use of the TRACE verb is in conjunction with an IF.

Army Standard: READY TRACE and RESET TRACE is only to be used as a debugging aid.

EXAMPLE:

IF IM01-ACCOUNT-NUMBER = 9546251
    READY TRACE.

This is used to TRACE beginning at a problem account number. Presumably processing up to this point has gone correctly and no TRACE was necessary. This saves paper and processing time.

TABLE HANDLING

A table is defined in the DATA DIVISION via the OCCURS clause. It consists of a series of elements identical in form.
EXAMPLE: A table of twenty-five names:

| WS-NAME(1) |
| WS-NAME(2) |
| WS-NAME(3) |
|           |
|           |
|           |
|           |
|           |

looks like: is defined by:

\[
\begin{align*}
\text{WS-NAME-TABLE.} \\
\text{WS-NAME} & \text{PIC X(20)} \\
\text{OCCURS 25 TIMES.}
\end{align*}
\]

This type of description is useful for storing a number of items that are the same size but differ in content. Once the description is complete, the individual item in the table may be referenced with either a subscript or index. A subscript or index simply tells the computer which one of the elements you want to look at or put something into.

EXAMPLE:

MOVE WS-NAME (1) TO WS-DETAIL-NAME.

The number in parentheses is a subscript. It indicates to COBOL that you are referencing the first of the twenty-five NAME entries. The thing in parentheses may also be a variable.

EXAMPLE:

77 WS-SUB PIC S9(2).

MOVE 1 TO WS-SUB.
MOVE WS-NAME (WS-SUB) TO WS-DETAIL-NAME.

This example also moves the first of the twenty-five NAME entries to NAME-OUT. In this case, since a subscript is used, the contents of the subscript are examined and are used to indicate which of the NAME entries are to be operated on. When using a subscript, it is important not to let the subscript get larger than the maximum number of entries in the table. If a table must be examined from beginning to end looking for a particular entry, the subscript may be varied in increments of one.
EXAMPLE (LINEAR STYLE):

MOVE 1 TO WS-SUB.
MOVE '0' TO WS-FLAG.
A. IF IM01-NAME EQUAL TO WS-NAME (WS-SUB)
   MOVE '1' TO WS-FLAG
   GO TO B.
   ADD 1 TO WS-SUB.
   IF WS-SUB LESS THAN 25
   GO TO A.
B. ADD 1 TO WS-CARD-COUNT.

This example initializes WS-SUB to 1 then searches the name table for a match with IM01-NAME. If a match is found, WS-FLAG is set to 1. If a match is not found after all the names have been examined, WS-FLAG is set to 0. Another way to do the same thing is to use the PERFORM VARYING verb.

EXAMPLE (STRUCTURED STYLE):

MOVE '0' TO WS-FLAG
PERFORM Ø100-NAME-COMPARE THRU Ø100-NAME-EXIT
   VARYING WS-SUB FROM 1 BY 1
   UNTIL WS-SUB IS GREATER THAN 25
   OR WS-FLAG = '1'
   ADD 1 TO WS-CARD-COUNT.
   ...
Ø100-NAME-COMPARE.
   IF IM01-NAME = WS-NAME (WS-SUB)
   MOVE '1' TO WS-FLAG.
Ø100-NAME-EXIT. EXIT.

This example sets WS-FLAG to '0' then PERFORMs Ø100-NAME-COMPARE through Ø100-NAME-EXIT. During the process of the PERFORM, WS-SUB is set to 1 and the conditions are tested. If the conditions are not true, then Ø100-NAME-COMPARE through Ø100-NAME-EXIT are executed. After Ø100-NAME-EXIT is finished, WS-SUB is incremented BY 1 and the conditions are tested. As soon as one of the conditions are true, the PERFORM statement is finished and the ADD 1 TO WS-CARD-COUNT is executed.

If the table consists of multiple entry items (e.g., name and social security number), the elementary items are referenced as though they are subscripted.
EXAMPLE:

01 WS-NAME-SSN-TABLE.
   05 WS-NAME-SSN OCCURS 25 TIMES.
      10 WS-NAME PIC X(20).
      10 WS-SSN PIC X(9).

   IF WS-NAME (WS-SUB) EQUAL TO IM01-NAME
       MOVE '1' TO WS-FLAG.

Even though the description of WS-NAME does not contain an OCCURS clause, WS-NAME is subordinate to WS-NAME-SSN which does contain an OCCURS clause. This being the case, WS-NAME must be referenced as though WS-NAME itself contained the OCCURS. Another optional entry with the OCCURS clause is INDEXED BY. This clause specifies an index to be used with the particular table defined by the rest of the description.

EXAMPLE:

01 WS-BAS-TABLE.
   05 WS-BAS-ENTRY OCCURS 30 TIMES
      INDEXED BY BAS-INDEX.
         10 WS-BAS PIC S999V99.
         10 WS-RANK PIC X(3).

BAS-INDEX is the index associated with the BAS-ENTRY table. WS-BAS and WS-RANK are referenced exactly as in using a subscript.

EXAMPLE:

MOVE WS-BAS (BAS-INDEX) TO WS-DETAIL-BAS.

To initialize an index to a value and decrement or increment, the index, the SET verb, is used.

EXAMPLES:

SET BAS-INDEX TO 1.
SET BAS-INDEX UP BY 2.
SET BAS-INDEX DOWN BY 1.

Indexes are used in conditions exactly like subscripts.

EXAMPLE:

IF BAS-INDEX GREATER THAN 25...

DRAFT ST 18-150
The linear example given at the beginning of the table handling section would look like this, using indexing instead of subscription, assuming that INDEXED BY NAME-INDEX is specified after the OCCURS clause.

**EXAMPLE:**

```
SET NAME-INDEX TO 1.
MOVE '0' TO WS-FLAG.
A. IF IM01-NAME EQUAL TO WS-NAME (NAME-INDEX)
   MOVE '1' TO WS-FLAG
   GO TO B.
   SET NAME-INDEX UP BY 1.
   IF NAME-INDEX LESS THAN 25
   GO TO A.
B. ADD 1 TO WS-CARD-COUNT.
```

The structured example would look like:

```
MOVE '0' TO WS-FLAG.
PERFORM 0100-NAME-COMPARE THRU 0100-NAME-EXIT
   VARYING NAME-INDEX FROM 1 BY 1
   UNTIL NAME-INDEX IS GREATER THAN 25
   OR WS-FLAG = '1'.
   ADD 1 TO WS-CARD-COUNT.

0100-NAME-COMPARE.
   IF IM01-NAME EQUAL TO WS-NAME (NAME-INDEX)
   MOVE '1' TO WS-FLAG.
0100-NAME-EXIT. EXIT.
```

Indexing is usually an order of magnitude more efficient than subscripting when stepping through a table from beginning to end. When using an index in a PERFORM VARYING statement, all of the SET statements are done automatically by the PERFORM.

88 entries cannot be used in or under an OCCURS.

**Army Standard:**

1. Indexing is more efficient than subscripting and is preferred.
2. Checks should be made to insure that indexes and subscripts do not exceed the range of the OCCURS.
3. Subscripting or indexing must be limited to three levels.
4. Subscripting or indexing beyond one level should be avoided.
DOS JOB CONTROL LANGUAGE (JCL)

// JOB name

This is the first card of each job. Name must be from 1 to 7 characters, the first of which must be alphabetic, the last of which may be characters or numbers. // is punched in columns 1 and 2.

// OPTION option-1, option-2, option-3...

This may appear after the JOB card and is used to change any of the standard system options. It is not required if the options which are standard are the options desired.

Options include:

   LINK      which specifies that the job is to be written on disk for link editing.
   LIST      which specifies that a listing of a compilation is to be written on SYSLLST.
   ERRS      specifies that a listing of any errors, if any, are to be listed on SYSLLST.
   DUMP      specifies that a hexadecimal dump of memory is to be made if the job is terminated abnormally.
   LISTX     specifies that an ALC listing of generated instructions is to be written on SYSLLST.
   SYM       specifies that a table of data-names are to be listed with their corresponding system-assigned names with locations.
   LOG       specifies that a listing of all job control is to be written on SYSLLST.
   XREF      specifies that a cross reference dictionary of data-names be written on SYSLLST.
   DECK      specifies that the compiler is to punch an object deck on SYSPCH.

The standard system options at the Institute of Administration are LIST, ERRS, NOLINK, NODUMP, NOLISTX, NOSYM, NOLOG, NOXREF, NODECK. Note that any option may be negated by preceding it with NO. If the programmer wanted to change one of the above options, such as NOLINK to LINK, he would punch the following card:

// OPTION LINK

If he wanted to change 2 items, such as NOLINK to LINK and LIST to NOLIST,

// OPTION LINK,NOLIST

NOTE: No space is permitted between a comma and the next option.
ASSGN SYSnnn,X'cuu'

The assign clause is used to assign a system logical number to a specific hardware device. 
nnn represents a 3 digit system logical number; cuu represents a channel and unit number to which a device is connected.

Devices include:
- card reader X'00C'
- card punch X'00D'
- printer X'00E'
- tape drives X'180' X'181' X'182' X'183' X'184' X'185'
- disk drives X'130' X'131' X'132' X'133' X'134' X'135' X'136'
- display (video) X'020' X'021'
- console typewriter X'01F'

System logical numbers on our system are available from 000 thru 029.

EXEC program-name

This statement causes a program on core-image library to be read into memory and executed. If no name is specified, the program compiled will be read into memory and executed.

EXAMPLE:

EXEC LNKEDT
would cause the link editor to be read into memory and link edit the program just compiled.

EXEC
would cause the program just link edited to be read into memory and executed.

/*

/* indicates end-of-data. It would appear after the last card of the source program to be compiled, for to the compiler the source program is data. It would also appear after any data which may have been included for use by an object program.

/&

& indicates end-of-job. When this card is encountered, the job is terminated. The JOB card of the next job should follow a /& card.
EXAMPLE of a job to be compiled:

// JOB SAMPLE
// ASSGN SYS004,X'132'
// EXEC FCOBOL
  IDENTIFICATION DIVISION.
  PROGRAM-ID. PE1.
    etc.
    rest of source program
  /
  /*
  /*

NOTE: The full (ANSI) COBOL compiler requires a work file on disk, and expects SYS004 to be assigned to a disk. Our disk file used for this purpose is X'132', and thus the ASSGN card shown above is required to compile ANSI COBOL programs.

EXAMPLE of a job to be compiled and executed with data included:

// JOB EXAMPLE
// OPTION LINK
// ASSGN SYS004,X'132'
// EXEC FCOBOL
Source deck
/*
  /* EXEC LNKEDT
  // ASSGN SYS005,X'0DC'
  // ASSGN SYS006,X'0DE'
  // EXEC
Card data
/*
/*

NOTE: Note that SYS005 must be assigned to a card reader in the COBOL source program since JCL has specified that SYS005 is to be assigned to X'0DC', a card reader. Likewise, SYS006 must be assigned to a printer in the COBOL source program.
There are three kinds of cards normally used in a typical OS job deck: the JOB card, the EXEC card, and the DD card. They will each be described to give an appreciation of the relationship to a COBOL program.

The JOB card has the following format:

```
//jobname JOB ... (other parameters)
```

This is always the first card in all JCL decks. It tells the operating system that a job is beginning.

The EXEC card has the following format when used in the context of a COBOL program being compiled, link-edited, and executed:

```
// EXEC COBULCG,PARM.COB=(compiler options),
// PARM.LKED=(linkage-editor options),
// COND.LKED=(n,LT),COND.GO=(n,LT)
```

The three cards shown, each beginning with a //, are treated as one card because the second and third cards are merely continuations of the first. A continuation is indicated by the card being continued ending with a comma and the next card beginning with a // followed by at least one space then the continued statement. In this case, the EXEC card is invoking a three step procedure (or PROC) that compiles a COBOL program, link-edits it, then executes that program. COBULCG is the name of the procedure. The first step in the procedure is named the COB step. PARM.COB refers to options that affect the execution of this step. Some of the commonly used options are:

- **QUOTE** indicates that a double quote is used to delineate literals instead of the apostrophe (or single quote) or vice versa.
- **APOST**
- **SPACE 1**
- **SPACE 2**
- **SPACE 3** indicate that the source program listing is to be single, double, or triple spaced respectively.
- **XREF** indicates that a cross reference is to be generated or suppressed of all of the data-names and paragraph-names after the source program listing.
- **NOXREF**
- **PMAP** indicates that a pseudo-assembler listing is to be printed or suppressed of the generated code in the PROCEDURE DIVISION after the source program listing and xref listing (if included).
- **NOMPAP**
- **DMAP** indicates that a map of the DATA DIVISION is to be printed or suppressed after the source program listing.
- **NODMAP**

If only one of the options is chosen, then the parentheses are not needed. If more than one option is chosen, the parentheses must be included and the options must be separated by commas.
The second step in the procedure is the LKED step. This is the step that link-edits the program. During this step, any subroutines that are used by the COBOL program are joined to it and a lot of busy work is done. There are two commonly used options in the linkage-editor.

MAP indicates that a map of the linked program is to be printed. This is useful in debugging the program using a core dump.

XREF indicates that a cross reference of the communication between the programs and subroutines is to be printed along with a map of the module.

The COND.LKED parameter is used to force execution of the LKED step even though diagnostics (errors) above the level of W have been generated during the compilation of the COBOL program. Normally a program will execute with C level errors but not E level or higher. To allow the program to link and execute with a C level error, substitute an 8 for the n in this parameter and in the COND.GO parameter. If linking and k execution is desired only with W level errors, then omit both COND parameters and the comma after the parentheses on the last PARM parameter.

After the EXEC card is the SYSIN card for the COBOL program. Its format is:

//COB.SYSIN DD *

COB indicates that it is to be used in the COB step; SYSIN is the name of the DD card; DD indicates that it is indeed a DD card (data description); and * indicates that this is a card file with the cards immediately following the DD card. The COBOL program is placed immediately behind this card.

After the last card in the COBOL program, the DD cards that describe the files used in the COBOL program. Their format is:

//GO.ddname DD ... (additional parameters)

The GO indicates that it is used during the GO step (execution); the ddname is identical to the external name useu in the system name after the ASSIGN in the FILE-CONTROL paragraph in the COBOL program; the DD indicates that this is a DD card; and the additional parameters can be many things. If this was to be a card file, then an asterisk (*) is all that is necessary after the DD. If this was a card punch file then SYSOUT=B is needed after the DD. If this was a printer file, then SYSOUT=A is needed after the DD. If the only entry after the DD is the word DUMMY and the file is used for sequential output (e.g., as a printer), everything written in that file is ignored by the system. If the DUMMY file is used as sequential input (e.g., as a card reader), the first read that is executed generates an end of file condition on that file. Random reads or writes are not allowed to DUMMY files.

If DISPLAY, EXHIBIT, or TRACE is used in the program, then the following card is needed.
EXAMPLE:

//EXAMINE JOB ... (other JOB card parameters)
//EXEC COBUCLG,
//   PARM.COB=(XREF,PMAP,DMAP,SPACE2),
//   PARM.LKED=XREF,
//   COND.LKED=(8,LT),
//   COND.GO=(8,LT)
//COB.SYSIN DU *
IDENTIFICATION DIVISION.
.
FILE-CONTROL.
   SELECT DISK-FILE
      ASSIGN TO UT-S-DISK.
   SELECT PRINTER
      ASSIGN TO UT-S-PRINTER.
   SELECT CARD-READER
      ASSIGN TO UT-S-READER.
   SELECT CARD-PUNCH
      ASSIGN TO UT-S-PUNCH.
.
   (rest of COBOL program)
.
//GO.DISK DD ... (other DD card parameters)
//GO.PRINTER DD SYSOUT=A
//GO.READER DD *
.
   (data cards)
.
//GO.PUNCH DD SYSOUT=B
//

In this example, the JOB name is EXAMPLE. The COBUCLG procedure is invoked. Parameters for the COB (COBOL compile) step are XREF, PMAP, DMAP, AND SPACE2. This will print a cross reference of all the data-names and paragraph names in the program, list all of the pseudo assembler generated codes in the procedure division, map the data division variables, and double space the COBOL source program listing. Parameters for the LKED (link-edit) step are XREF. This will map the linked module and cross reference the communication between the routines. The COND parameters indicate that linking and execution is desired with C level errors in compilation. The COB.SYSIN card indicates that the program immediately follows on cards.

The four files that the program uses are defined to have external names of DISK, PRINTER, READER, and PUNCH. These same names are all defined as DD names beneath the COBOL program. The GO.DISK DD card describes the disk file. The GO.PRINTER DD card indicates that this is a printer file by having SYSOUT=A after the Du. The GO.READER DD card indicates that this is a card file by the
asterisk (*) after the DD. The GO.PUNCH DD card indicates that this is a card punch file by the SYSOUT=B after the DD. Finally the last card in the deck is a // with nothing else on the card. This indicates the end of the job.
COBOL SOURCE DESIGN LANGUAGE (SDL)

1. ADD: Adds the contents of one data name to another data name or a numeric integer value to a data name.

2. AT END: Used only in conjunction with the READ.

3. CLOSE: Makes a file nonaccessible, the file can no longer be read from or written to.

4. EXIT: Used to identify the ending point of a routine.

5. IF-THEN-ELSE: The IF portion states a condition, the THEN portion is done when condition is true, the ELSE portion is done when the condition is not true.

6. MOVE: Moves the contents of one data name to another data name.

7. OPEN: Makes a file accessible so that it can be read from or written to.

8. PERFORM: Does a routine from its beginning point through its ending point, one time.

9. PERFORM-UNTIL: Does a routine from its beginning point through its ending point, some number of times until a stated condition is true.

10. READ: Accesses a file and makes one record available for processing.

11. STOP RUN: Used to show the end of all processing. The STOP RUN will be used at the end of the main controlling routine.

12. WRITE: Puts the contents of a record name out to a physical device.

13. NEXT SENTENCE: Used in conjunction with the IF-THEN-ELSE to show a null THEN or ELSE portion of the statement.
The Programming Standards established here are not intended to be all-inclusive for ANS-COBOL program development. Rather, they are presented in order to eliminate ambiguity in the completion of COBOL practical exercises and tests by students of Computer Science Department. Standards already established by US Army Computer Systems Command are included in this memorandum. It should be noted that the standards specified are mandatory for graded practical exercises and tests administered by the Software Division. Specifications are grouped by COBOL Division, i.e., Section I deals with requirements in the Identification Division, Section II with Environment Division, Section III with Data Division and Section IV with Procedure Division. Where these standards conflict with ST 18-150, these standards have precedence.

Section I. IDENTIFICATION DIVISION

Though the only entry required by the COBOL compiler is that of a Program-ID, all paragraphs must be included in every program.

Requirements:

a. Program-ID. A meaningful name must be provided for this entry.

b. Author. The student will enter his Rank, Name and Class Number.

c. Installation: Enter

   CSD, USAIA, Ft Harrison, IN 46216

d. Date-Written: The date the student started coding the program will be entered here.

e. Date-Compiled. Enter TODAY.

f. Security. Enter UNCLASSIFIED.
g. Remarks. This paragraph will be used to provide a description of the functions the program performs. It will also include an explanation of the use of all level 77 and all level 01 entries in working-storage. An asterisk (*) will be placed in column 7 of each line (card) of this paragraph, including the paragraph header line.

Example of Identification Division entries:

<table>
<thead>
<tr>
<th>IDENTIFICATION DIVISION.</th>
<th>PROGRAM-ID.</th>
<th>D-_STANDARD-EXAMPLE.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>CPT I M A STUDENT</td>
<td>ADPC 3-77.</td>
</tr>
<tr>
<td>INSTALLATION.</td>
<td>CSD USAF</td>
<td>FTHARRISON, INN 6216.</td>
</tr>
<tr>
<td>DATE-WRITTEN.</td>
<td>17 JAN 77.</td>
<td></td>
</tr>
<tr>
<td>DATE-COMPILTED.</td>
<td>TODAY.</td>
<td></td>
</tr>
<tr>
<td>SECURITY.</td>
<td>UNCLASSIFIED.</td>
<td></td>
</tr>
</tbody>
</table>

*REMARKS. THIS PROGRAM PRODUCES A LISTING OF PERSONNEL IN GRADE OF PV1 ASSIGNED TO FORT HARRISON. EACH SOLDIER'S NAME, SSN AND UNIT OF ASSIGNMENT ARE PRINTED.*

**DESCRIPTION OF 77'S.**

**WS-EOF-SW.** THIS IS A 3 CHARACTER FIELD INITIALIZED TO 'OFF'. IT IS USED TO CONTROL THE READ-AND-PROCESS-LOOP. AT END OF INPUT FILE ITS VALUE IS CHANGED TO 'ON'.

**DESCRIPTION OF WORKING-STORAGE 01'S.**

**WS-HEADING-LINE.** CONTAINS CONSTANT DATA TO BE PRINTED AT TOP OF EVERY PAGE OF THE LISTING.

**WS-DETAIL-LINE.** WILL CONTAIN DATA PERTAINING TO EACH SOLDIER.
Section II. ENVIRONMENT DIVISION

Since student PE's are run on the IBM 360-40 under QS-MFT, the requirements for this section are oriented to that system.

a. In the Configuration Section both the Source-Computer and Object-Computer paragraphs will contain the entry: IBM-360-H40. The Special-Names paragraph will be used to equate the name "TOP-OF-PAGE" to the printer carriage control tape channel 1. Example of Configuration Section entries:

```
Configuration Section,
SOURCE-COMPUTER. IBM-360-H40.
OBJECT-COMPUTER. IBM-360-H40.
SPECIAL-NAME. COL IS TOP-OF-PAGE.
```

b. Input-Output Section. The particular requirements for this section are contained in each practical exercise (PE), except for the formation of the individual file-name. The following standards will be used in forming file-names.

1. The first character of the name must be I, indicating that the file is an input file in this program, or O, indicating that the file is an output file, or B, indicating that the file is used for both input and output. For sort files, the first character will be S.

2. The second character will be selected from the following:
   - M - used as Master Files
   - T - used as Transaction Files
   - R - used as Report Files written to the printer
   - C - used for Card Files which contain miscellaneous data, e.g., data to build a table, control data to extract data from other files
   - W - Work (intermediate) Files

3. The third character will always be a hyphen (-).

4. The rest of the file-name must be descriptive of what the file contains.

IA-01-02-05
(5) Examples:

(a) A personnel master file is used as input to a program.

(b) A printed listing of personnel data is produced by a program. It is an output file.

(c) A supply parts master file, organized as an indexed sequential file, is read from the written to:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(2) SELECT IA-PERS-MASTER
    ASSIGN TO UT-S-SYS#06.

(3) SELECT OR-PERS-LISTING
    ASSIGN TO UT-S-SYS#05.

(4) SELECT BM-PARTS
    ASSIGN TO DA-I-SYS#08.

(6) The ASSIGN Clause of the SELECT statement will be indented four (4) columns and punched on a separate card.
Section III. DATA DIVISION

(a) File Section.

(1) Though the only clause required by the compiler in a File Description (FD) is the LABEL RECORD clause, all optional entries will be included for all files. Each clause will be punched on a separate card and will begin in column 12. Example of an FD:

```
FD  IM-PERS-MASTER
LABEL RECORD IS STANDARD,
BLOCK CONTAINS 1 RECORDS,
RECORD CONTAINS 64 CHARACTERS,
DATA RECORD IS IM01-PERS-MSTR.
```

The blocking factor and record size are specified on the record layouts for each PE. The data record name is a programmer-defined name and will be formed according to these standards:

(a) The first two characters will be identical to the first two characters of the file-name.

(b) The next two characters will consist of a sequence number beginning with 01. If there is only one record format associated with a file, that record name will have 01 as the third and fourth characters. If more than one record format, the first one described will have a sequence number of 01, the second one 02, and so on.

(c) The fifth character will always be a hyphen (-).

(d) The rest of the data record name will be descriptive of the record itself.

(e) If a record is defined as an elementary item, the picture clause will begin in column 48 (or as specified in para b(4)) and the word PICTURE will be abbreviated to PIC.
(2) Standards for defining data items within records are:

(a) Level numbers will begin at 05, and each subordinate level will be incremented by 5. Subordinate elements will be indented four (4) columns.

(b) The first five characters of all data item names will be identical to the first five characters of the record name. Allow two blank columns between level numbers and the data names.

(c) The rest of the data item name will be descriptive of the data item.
The following is an example of the description of the record associated with IM-PERS-MSTR, the file defined in paragraph a(1) of this section.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>IMOL-PERS-MSTR.</td>
</tr>
<tr>
<td>05</td>
<td>IMOL-NAME.</td>
</tr>
<tr>
<td>10</td>
<td>IMOL-LAST-NAME</td>
</tr>
<tr>
<td>10</td>
<td>IMOL-FIRST-NAME</td>
</tr>
<tr>
<td>05</td>
<td>IMOL-SSN</td>
</tr>
<tr>
<td>05</td>
<td>IMOL-RANK</td>
</tr>
<tr>
<td>05</td>
<td>IMOL-DATE-OF-RANK</td>
</tr>
<tr>
<td>05</td>
<td>IMOL-NR-DEPS</td>
</tr>
<tr>
<td>05</td>
<td>IMOL-DATE-OF-BIRTH</td>
</tr>
<tr>
<td>10</td>
<td>IMOL-DAY-OF-BIRTH</td>
</tr>
<tr>
<td>10</td>
<td>IMOL-MON-OF-BIRTH</td>
</tr>
<tr>
<td>10</td>
<td>IMOL-YR-OF-BIRTH</td>
</tr>
<tr>
<td>05</td>
<td>IMOL-NTD-GROSS-PAY</td>
</tr>
<tr>
<td>05</td>
<td>FILLER</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(d) The PICTURE clause for elementary items will begin in column 48 (or as specified in para b(4)). The word PICTURE will be abbreviated to PIC.

(e) Each additional clause will be punched on a separate card, and will begin in column 28.

b. Working-Storage Section.

(1) The first entry in Working-Storage will be a level 77 item indicating the beginning of the section, and the last entry will be an 01 item indicating the end of the section. Insertion of these two items aids in debugging with a core dump. (See example on page 10.)

(2) All data elements defined in WORKING-STORAGE will be preceded by characters (WS-) as the first three characters of the name.

(3) The rest of the name will be formed as follows:

(a) For level 77 and 01 items, it is sufficient to provide a descriptive name. Examples:

To name a page counter using a level 77:

```
77  WS-PAGE-COUNTER
```

To name an independent record for a printed detail line using a level 01:

```
01  WS-DETAIL-LINE
```

(b) All data elements subordinate to a level 01 item will contain in positions 4 and 5 (and occasionally 6) of the name, an identifier which links them to the level 01 item: Examples: A field to be printed as part of a detail line and which contains a person's social security number could be named as: WS-HL1-SSN

A field to contain a date and be printed as part of Heading Line One could be named as: WS-HL1-DATE

(c) Rules for level numbers and indentation are the same as outlined for items in the file section. See para a(2)(a) of this section.

IA-01-02-05
(4) Any clauses included with a date item description will be in the following order:

REDEFINES
OCCURS
PIC
USAGE
VALUE
JUSTIFIED
SYNC

(5) The first clause will be punched in column 48, and each additional clause will be punched on a separate card beginning in column 28.

(6) All numeric level 77 items will be assigned an initial value.

(7) The following example of a WORKING-STOREAGE SECTION is provided to clarify the standards outlined above.
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>WORKING-STORAGE SECTION.</td>
<td></td>
</tr>
<tr>
<td>WS-BEGIN-POINT</td>
<td>VALUE &quot;WORKING STORAGE BEGINS HERE&quot;.</td>
</tr>
<tr>
<td>WS-LINE-COUNTER</td>
<td>USAGE COMP.</td>
</tr>
<tr>
<td>WS-EOF-SWITCH</td>
<td>VALUE &quot;OFF&quot;.</td>
</tr>
<tr>
<td>WS-DETAIL-LINE</td>
<td></td>
</tr>
<tr>
<td>FILLER</td>
<td>VALUE SPACES.</td>
</tr>
<tr>
<td>WS-DL-NAME</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>WS-DL-FIRST-NAME</td>
<td>PIC X(12)</td>
</tr>
<tr>
<td>WS-DL-LAST-NAME</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>WS-DL-SSN</td>
<td>PIC X(18)</td>
</tr>
<tr>
<td>WS-END-POINT</td>
<td>VALUE 'WORKING STORAGE ENDS HERE'.</td>
</tr>
</tbody>
</table>
Section IV. PROCEDURE DIVISION

a. PARAGRAPH Names:

(1) All paragraph names will contain in the first four positions a sequence number, beginning with 0010. The number assigned to each paragraph will be in ascending order and each will be incremented by 10 to allow insertion of additional paragraphs as the need arises. An exception will be numbers assigned to EXIT paragraphs. These will contain the same number as the preceding paragraph. All paragraphs will have an associated exit paragraph, except, of course, exit paragraphs themselves.

(2) The fifth position of the name will be a hyphen (-), and the remainder of the name will be descriptive of the functions(s) performed.

(3) Example:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>12 16 20 24 28 32 36 40 44 48</td>
</tr>
<tr>
<td>0010</td>
<td>MAIN-DRIVER</td>
</tr>
<tr>
<td>OPEN</td>
<td>PERFORM READ-PRINT UNTIL WS-IM-EDF-SW EQUAL 'ON'</td>
</tr>
<tr>
<td>CLOSE</td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td>EXIT</td>
</tr>
<tr>
<td>STOP RUN</td>
<td></td>
</tr>
<tr>
<td>0024</td>
<td>READ-PRINT</td>
</tr>
<tr>
<td>READ</td>
<td>MOVE</td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td>EXIT</td>
</tr>
<tr>
<td>EXIT</td>
<td></td>
</tr>
</tbody>
</table>

b. Verbs: Any reference to indentation implies four (4) columns.
(1) **PERFORM.** The "Thru" option will always be used with a **PERFORM** verb. Procedure - name - 2, which follows the word **THRU**, must be an exit paragraph. Example:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>PERFORM 0020-READ-PRINT THRU 0030-EXIT.</td>
<td></td>
</tr>
</tbody>
</table>

**0030-READ-PRINT.**

READ IM-PERS-MASTER

AT END

MOVE 'ON' TO WS-IM-EOF-SWITCH.

WRITE ----

**0030-EXIT.**

----

(a) When the "UNTIL" option is used with a **PERFORM**, the UNTIL clause will be indented and appear on a separate line. Example:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>PERFORM 0020-READ-PRINT THRU 0030-EXIT</td>
<td></td>
</tr>
</tbody>
</table>

UNTIL WS-IM-EOF-SW EQUAL 'ON'.

(b) When using the "VARYING" and "UNTIL" options together, each clause will be indented and appear on a separate line from the **PERFORM**. Example:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>PERFORM 0020-TABLE-BUILD THRU 0090-EXIT</td>
<td></td>
</tr>
</tbody>
</table>

VARYING WS-SUB FROM 1 BY 1

UNTIL WS-SUB GREATER 36.
(2) **IF.**

(a) All IF statements will include the optional word THEN following the condition. The word THEN will be indented and appear on the next line. If the statement following the THEN is another IF (i.e., a nested IF), it will also be indented and appear on a separate line. The same rules apply to the use of the ELSE option.

(b) Examples: See Page 14.

(3) **Go To.** A GO TO statement may only be used to transfer control to a section exit or a paragraph exit.

(4) **SORT.** All clauses specified with the SORT verb will be indented and each clause will appear on a separate line. When either or both of the "INPUT PROCEDURE IS" and "OUTPUT PROCEDURE IS" options are used, each of the sections referred to must be clearly identifiable as a separate section. A work section will be used as the last section of the program, and it will contain individual routines (with appropriate exists) which are referenced thru the use of PERFORM statements in the INPUT PROCEDURE section or OUTPUT PROCEDURE section. Example: See Page 15.

(5) **READ.** AT END and INVALID KEY conditions associated with READ statements will be indented and appear on a separate line. Only one READ statement will be used for each input file.
### IF Statement EXAMPLES

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>14</td>
</tr>
</tbody>
</table>

**IF** IM01-YEAR-OF-BIRTH LESS THAN 48
**THEN** PERFORM 0180-PT-TEST-ROSTER THRU 0080-EXIT.

**IF** IM01-YEAR-OF-BIRTH LESS THAN 48
**THEN**
**IF** IM01-MARITAL-STATUS EQUAL 'N'
**THEN** PERFORM 0040-PRINT THRU 0040-EXIT
**ELSE** PERFORM 0120-ADD THRU 0120-EXIT
**ELSE**
**IF** IM01-PT-SCORE LESS THAN 300
**THEN** PERFORM 0110-PT-FAIL THRU 0110-EXIT
**ELSE** GO TO 0060-EXIT.

*Note: Corresponding "thens" and "else's" are aligned.*
**SORT Procedure EXAMPLES**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>24</td>
<td>28</td>
</tr>
<tr>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>40</td>
<td>44</td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
</tbody>
</table>

SORT SW-SORT-WORK-FILE
ON ASCENDING KEY SW-1-PART-NUMBER
INPUT PROCEDURE IS $\phi 30$-IN-TO-SORT
OUTPUT PROCEDURE IS $\phi 30$-SORT-OUT.

$\phi 30$-IN-TO-SORT SECTION.
OPEN INPUT IM-PARTS-MASTER.
PERFORM $\phi 110$-READ-RELEASE THRU $\phi 110$-EXIT
UNTIL IM-EOF-SW EQUAL 'ON',
CLOSE IM-PARTS-MASTER.

$\phi 30$-SECTION-EXIT.
EXIT.

$\phi 30$-SORT-OUT SECTION.
OPEN OUTPUT OR-LISTING
PERFORM $\phi 120$-RETURN-PRINT THRU $\phi 120$-EXIT
UNTIL WS-SW-EOF-SWITCH EQUAL 'ON',
CLOSE OR-LISTING.

$\phi 30$-SECTION-EXIT.
EXIT.

$\phi 40$-WORK SECTION.
$\phi 110$-READ-RELEASE.
READ IM-PARTS-MASTER
 AT END --------

$\phi 10$-EXIT.
EXIT.

$\phi 120$-RETURN-PRINT.
-- -- -- -- -- --

$\phi 120$-EXIT.
EXIT.
(6) WRITE. Only one WRITE statement per record type is permitted. When writing to the printer, both the FROM and AFTER ADVANCING options may be used. The AFTER ADVANCING clause will be indented and appear on a separate line.

(7) The following verbs will not be used:

MOVE Corresponding
ALTER
GO TO DEPENDING ON
ENTER
ON

(8) Miscellaneous. All switches will be set up as three characters alphanumeric fields. The words "OFF" and "ON" will be used to change the status of a switch, or as a literal to check for a specific value in a switch.
1. INTRODUCTION.
   
a. ANSI COBOL, with its IBM extensions, has many advanced processing capabilities; that is, it has features that are not really needed to write most programs but can make it easier to write the programs, or make the programs themselves more flexible. Among those items which might be considered advanced processing are the following:

   1) The segmentation feature, which allows programs to be divided into segments.
   
   2) The source program library, which allows parts of programs to be saved in a library for later use.
   
   3) The debugging feature, which allows for a diagnostic printout while the program is executing.
   
   4) The report writer feature, which provides an alternate method of describing printed output files.
   
   5) The sort feature, which permits rearranging records of a file into any order.
   
   6) Index-sequential file handling, which allows processing of an indexed-sequential direct-access file.

   b. We do not have the time nor space to discuss all these features, nor do all of them merit discussion. We will cover the two that seem to be used most often: the sort feature and index-sequential file handling. The others we will leave for you to explore on your own.

2. OBJECTIVE. You will learn the COBOL statements necessary to create, read sequentially, and update randomly an index-sequential file. You will also learn the statements and methods needed to use the sort feature. (You should note that index-sequential files are an IBM addition to the standard COBOL language. Although many other manufacturers also have adopted index-sequential files, the system with which you are working may not have this feature.) The two IBM operating systems in general used by the Army are DOS (Disk Operating System) and OS (Operating System). Where one or the other operating system has unique requirements, they are identified.

3. THE COBOL SORT FEATURE. The COBOL sort feature enables you to rearrange a file used by your program into any order you want. You can do the sorting as part of your program, instead of having to use a separate utility. If you have a file in social security number order, for example, you can resort it into alphabetical order by name, or by address or into whatever order you would like. To use the sort, we must set up a sort-file in our program. The purpose of a sort-file is to gather up all the records to be sorted; to hold them while the sort takes place, and to parcel the records out once they have been rearranged. The sort-file is essentially a work area into which the records to be sorted are placed. One method of sorting consists of three separate phases; each is described below.
a. Phase 1 - Input. The records to be sorted are read into the computer's memory, and "strings" of records are built and written to the sort-file. A "string" consists of a series of records in the desired sequence. As soon as the sequence is broken, a "string" is complete, and a new one is started. For example, let us say that records 4, 7, 8, 10, and 13 have been read and are put into a "string". The next record read is 9. The sequence has been broken, and a completed "string" of records 4, 7, 8, 10, 13 is now on the sort-file. A new "string" with record 9 as the first record is now started.

b. Phase 2 - Sort/Merge. After all records have been read and placed in "strings", two "strings" are brought back into memory at a time, and merged into one longer "string". This continues until only two "strings" are left.

c. Phase 3 - Output. The last two "strings" are merged when the records are passed back to memory or to an output device - whichever the program calls for.

Note: The number of "strings" which are merged at one time depends on what is known as the "merge order". The "merge order" is a function of the amount of memory available, number and type of input/output devices and number of work files to be used.

2. When using the COBOL SORT feature, input to and output from the SORT can be handled in one of several ways. First, let us look at what a programmer can do in the input phase. Basically, he has two options. They are:

   a. Copy every field of every record in the input file to the sort-file.

   b. Copy only selected fields and/or records in the input file to the sort-file.

   If option "a" is selected, the input file is copied in its entirety. This method of input is useful when a file is simply to be rearranged, and all records are required for output. If option "b" is selected, only those records which need to be sorted are passed to the sort file. Also, individual fields which are not needed on output can be dropped, thereby making the sort record smaller. Removal of unnecessary records and fields reduces the amount of data to be sorted, and allows for better use of available memory. The time required for the sort itself will decrease. This option is normally used when printing a report from a file and

   a. the report must be in a different order than the file,

   b. only certain records are to be included and

   c. not all fields on the original record are needed on the report.
6. Output may be one of two options:
   a. Copy the sorted file to a permanent file;
   b. Read the sorted file as input and process it, one record at a time, like any other sequential input file.

Note: Option "a" is used when a permanent copy of the sorted file is desired and no reports or special processing are required. Option "b" is used when reports and/or special processing are required.

7. DESCRIBING THE SORT-FILE (Sort Work Area). Like any other file, we must have a SELECT statement for the sort-file. However, instead of having a corresponding FD in the Data Division, we must use a Sort Description (SD). Let us look at the SELECT first.

   a. SELECT for a sort-file under IBM Disk Operating System (DOS) COBOL. The file must be either a tape or a disk file. It is divided into sections, each section on a different tape or disk area. The more sections into which the file is broken, the faster the sort will execute. If you are using disk as your work unit, you may have from one to eight sections. If you are using tape, you may have from three to nine. You may not mix disk with tapes. The choice of how many work units to use will depend mostly on how many tapes or disk areas are available. The format of the SELECT clause may be found in the IBM ANS COBOL Language Reference Manual. An example of a SELECT for a sort-file that will use four work units on disk is SELECT SORTING, ASSIGN TO 4 SYS001-UT-2314-S-SORTWK1. Note that the ASSIGN clause for the sort-file must specify SYS001. The unit may be either 2400 (tape) or 2314 or 2311 (disk). The -SORTWK1 parameter must be included when you are using standard labels. An example of a SELECT statement for a sort-file under OS is:

   SELECT SORT-WORK-FILE ASSIGN TO 4 UT-S-SORTWK1

   b. The sort description. In a program that uses a sort, each ordinary file must have an FD and the sort-file must have an SD. The format for the SD can be found in the IBM ANS COBOL Language Reference Manual under Sort File Description Entry.

Note: The sort-file-name must match the one given in the SELECT for the sort file, and the BLOCK CONTAINS clause may NOT be coded or used with an SD.

8. SORTING THE FILE. The file is sorted by executing a SORT statement in the PROCEDURE DIVISION. Each time the SORT is executed, the entire file is sorted. Usually, the SORT will be executed once. The SORT statement provides information about all phases of the sorting operation. The format of the SORT statement is shown below. The sort-file itself is never opened or closed by the programmer. Opening and closing of the sort-file is done automatically by the SORT verb. For the format of the SORT statement, refer to the IBM ANS COBOL Language Reference Manual. Let us see how each part of the SORT statement applies to a particular phase of the SORT operation.
a. Phase 1--Input. The INPUT PROCEDURE or USING clause (not both) determines how the sort-file will get the records that it is to sort.

(1) If the entire input file is to be sorted, you can use the USING file-name-2 option. This will cause the file-name-2 file to be copied onto the sort-file. You must not open, close, or read the file-name-2 file; copying is done automatically. (NOTE: file-name-2 cannot be open while the sort is executing.) When you code the USING, you do not have access to the records as they are being copied onto the sort-file. The record description under the SD must be identical in form to the record description under file-name-2.

(2) If you wish to manipulate the records from the input file before they are placed on the sort-file, you must use the INPUT PROCEDURE clause. By coding the INPUT PROCEDURE clause, you will cause the input phase to perform the section-name you give in the INPUT PROCEDURE clause.

(3) A section is made up of one or more paragraphs that are logically related. Every section is begun with a section name (a paragraph name followed by a space and the word SECTION). A section ends on the statement before the next section name or with the last card in the program, whichever occurs first. It is a good idea to have an exit paragraph (called section-exit in this text) at the end of each section. This allows the structured construction of the procedure division to be maintained.

(4) When the sort begins execution, the input procedure section is executed. The input procedure section is actually a self-contained program that:

(a) opens the file that has the records to be sorted;
(b) reads those records;
(c) processes them in whatever way is needed;
(d) passes the records to the sort file by using a RELEASE statement (explained later);
(e) closes the input file;
(f) and branches to the section exit.

Note: The input procedure section has its own driver paragraph (ending with a GO TO to the section-exit), any routines needed for processing the input records, and a section exit at the end of all the routines.

b. Phase 2--Sort/Merge. The SORT...ON...KEY portion of the sort statement controls the action of this phase. Once Phase 1 is finished, all the records that are to be sorted will be arranged in strings in the sort-work area. The SORT...ON...KEY will determine how the strings are to be merged. Rules on sort keys are listed below. If the file was in SSN order and we wish to sort it alphabetically by name, then NAME is the key on which we will be sorting. There can be more than one sort key; for example, we might want to sort the records by STATE and then by CITY within each STATE. In this case the STATE field would be the major sort key, and the CITY field would be the minor sort key. When you code the ON...KEY clauses, you must code the major key first, followed by the minor keys. For each key, you have the choice of sorting it...
in ASCENDING (lowest first, highest last) sequence or DESCENDING (highest first, lowest last) sequence. (An alphabetical sort would be ASCENDING - A is first, Z is last). Rules on keys on DOS and OS:

1. The sort keys that you are sorting on must be subdivisions of the record of the sort-file; i.e., they must come under the 01 which belongs to the SD.

2. The maximum number of keys is 12.

3. The total length of the keys must not exceed 256 bytes.

4. Keys cannot contain or be subordinate to an OCCURS clause.

5. Keys cannot be located after an OCCURS clause.

6. All keys must be defined in the first 4092 bytes of the sort record.

c. Phase 3--Output. Once Phase 2 is finished, the records are in a sort work area and can be retrieved in the order in which you want them. The means of retrieving them is specified by the GIVING or OUTPUT PROCEDURE clauses.

1. The GIVING file-name-3 operates in a similar manner to the USING; i.e., the sort-file is copied to an output file. The file that receives the sorted records must not be opened by the programer when the sort is executed. File-name-3 will be opened and closed automatically. As with the USING, you do not have access to the records as they are copied from the sort-file to file-name-3. The record description of file-name-3 must be identical in form to the record description in the sort-file.

2. By using the OUTPUT PROCEDURE clause, you may retrieve the sorted records one at a time and process them. Like the input procedure, the section-name you give will be performed. Within the section you can retrieve records from the sort-file by using a RETURN statement (discussed later in this text). The sort file must not be opened or closed. As in the input procedure section, the output procedure section is responsible for opening and closing the output file(s), processing the records returned from the sort, and branching to its section exit when all processing is complete. It should begin with a driver routine, be followed by all the routines needed to process the sorted records and end with its section-exit paragraph. Once the performance of the OUTPUT PROCEDURE is finished, phase 3 of the sort operation is done; and control is passed to the sentence following the sort verb.

9. THE RELEASE STATEMENT. The purpose of an INPUT PROCEDURE is to let you process the records before they are passed on to the sort-file. Once you have finished processing an input record in your INPUT PROCEDURE, you must use the RELEASE statement to place the record on the sort-file. The format of the RELEASE can be found in the IBM ANSI COBOL Language Reference Manual. The RELEASE is the equivalent of a WRITE statement for use on sort-files. Even
the FROM option works in the same way as it does for a WRITE; i.e., if the FROM is used, the data-name indicated in the FROM option is moved to the sort-record area and then the sort-record is RELEASED. The RELEASE statement writes the contents of the sort-record onto the sort-file. It may only be used within the bounds of the section named in the INPUT PROCEDURE clause.

10. THE RETURN STATEMENT. The purpose of an OUTPUT PROCEDURE is to let you process the records after they have been sorted. The RETURN statement may only be used within the bounds of the section named in the OUTPUT PROCEDURE clause. If the RELEASE is the equivalent of a WRITE for a sort file, then the RETURN is the equivalent of a READ for a sort-file. The format of the RETURN is found in the IBM ANS COBOL Language Reference Manual. Where the RELEASE called for the name of the sort records, the RETURN calls for the name of the sort file. The INTO option and AT END clause operate in the same way as for a READ. If the INTO option is used, the record description of the output file must be identical in form to the record description of the sort file. The AT END condition will occur when all the records of the sort file have been returned. The statements between the words AT END and the period that ends the sentence will not be executed until the AT END condition occurs.

11. SORT-WORK SECTION. A WORK section following the input and output procedure sections is used to contain all the routines used in the program. The input and output procedures then PERFORM routines contained in the SORT-WORK SECTION. The SORT-WORK SECTION is used not only to preserve a structured program design, but also to define the end of the input and output procedure sections.

12. A sample program which uses the SORT verb is at Inclosure 1.
PRACTICAL EXERCISE

Personnel Listing

1. The Personnel Office has requested a listing of all personnel in the battalion who have completed more than 35 months of service.

2. You are provided with a report format and card layout.

3. The report will be generated from the personnel master file. The file is stored in card-image form on a 2314 disk file.

4. You will keep a count of the total number of people who have more than 35 months of service. In addition, only 20 lines are to be printed (assuming double spacing) on a page. If more than 20 lines need to be printed start over with a new page.

---

FORM LAYOUT

```
1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
NAME        RANK    SSN      MON SER
 XXXXXXXXXXXXXXXXXXXXXXXXXX XXXX XXXXXXXXXXXXXX XXX
 XXXXXXXXXXXXXXXXXXXXXXXXXX XXXX XXXXXXXXXXXXXX XXX
 XXXXXXXXXXXXXXXXXXXXXXXXXX XXXX XXXXXXXXXXXXXX XXX
TOTAL NUMBER OF PERSONNEL WITH OVER 35 MONTHS SERVICE XXX
```

---

PERSONNEL MASTER INPUT FILE

<table>
<thead>
<tr>
<th>Record Position</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 2</td>
<td>Battalion</td>
</tr>
<tr>
<td>3</td>
<td>Company</td>
</tr>
<tr>
<td>4 - 12</td>
<td>Social Security Number</td>
</tr>
<tr>
<td>13 - 30</td>
<td>Name</td>
</tr>
<tr>
<td>31 - 32</td>
<td>Pay Grade</td>
</tr>
<tr>
<td>33 - 35</td>
<td>Rank</td>
</tr>
<tr>
<td>36 - 41</td>
<td>ETS</td>
</tr>
<tr>
<td>42 - 47</td>
<td>Date of Entry</td>
</tr>
<tr>
<td>48 - 50</td>
<td>Months of Service</td>
</tr>
<tr>
<td>51 - 52</td>
<td>Number of Dependents</td>
</tr>
<tr>
<td>53 - 80</td>
<td>Not Used</td>
</tr>
<tr>
<td></td>
<td><strong>DUMP DEBUG WORKSHEET</strong></td>
</tr>
<tr>
<td>---</td>
<td>-------------------------</td>
</tr>
<tr>
<td>1.</td>
<td>Hexadecimal Memory Location of Interrupt</td>
</tr>
<tr>
<td>2.</td>
<td>Load Point</td>
</tr>
<tr>
<td>3.</td>
<td>Relative Interrupt Address</td>
</tr>
<tr>
<td>4.</td>
<td>ALC Instruction Interrupted</td>
</tr>
<tr>
<td>5.</td>
<td>Corresponding COBOL Statement Number</td>
</tr>
<tr>
<td>6.</td>
<td>COBOL Statement</td>
</tr>
<tr>
<td>7.</td>
<td>COBOL Source Data Names</td>
</tr>
<tr>
<td></td>
<td>Receiving Operand</td>
</tr>
<tr>
<td></td>
<td>Sending Operand</td>
</tr>
<tr>
<td>8.</td>
<td>Internal Data Name for</td>
</tr>
<tr>
<td></td>
<td>Receiving Operand</td>
</tr>
<tr>
<td></td>
<td>Sending Operand</td>
</tr>
<tr>
<td>9.</td>
<td>Data Format of</td>
</tr>
<tr>
<td></td>
<td>Receiving Operand</td>
</tr>
<tr>
<td></td>
<td>Sending Operand</td>
</tr>
<tr>
<td>10.</td>
<td>Length in bytes reserved by</td>
</tr>
<tr>
<td></td>
<td>Receiving Operand</td>
</tr>
<tr>
<td></td>
<td>Sending Operand</td>
</tr>
<tr>
<td>11.</td>
<td>Machine Instruction Interrupted</td>
</tr>
<tr>
<td>12.</td>
<td>Operation Code of Machine Instruction</td>
</tr>
<tr>
<td>13.</td>
<td>Machine Instruction Format</td>
</tr>
<tr>
<td>14.</td>
<td>Length of the Machine Instruction in bytes</td>
</tr>
<tr>
<td>15.</td>
<td>How many Operand Lengths in Machine Instruction</td>
</tr>
<tr>
<td>16.</td>
<td>What is/are the Length(s)</td>
</tr>
</tbody>
</table>

17. Base Register for First Operand

18. Base Address for First Operand

19. Displacement for First Operand

20. Absolute Address of First Operand

21. Contents of Storage Allocated to First Operand

22. Base Register for Second Operand

23. Base Address for Second Operand

24. Displacement for Second Operand

25. Absolute Address of Second Operand

26. Contents of Storage Allocated to Second Operand

IF TEMPORARY STORAGE IS REFERENCED IN THE ALC INSTRUCTION (TS= in LISTX)
THEN FILL IN ITEMS 27 THRU 30.

27. Relative Address of TS

28. Absolute Address of TS

29. Internal Data Name of Item in Left Half of TS

30. Internal Data Name of Item in Right Half of TS

31. Origin of Data and /or Reason for Incorrect Format:
   a. Input Record _____  b. Failure to Initialize _____
   c. Failure to Generate _____

32. Corrective Action:

121-013-1413-150-A

2

IA-01-01-21

USAFPP 171-19 6/79
DEBUG
USAIA
SOFTWARE DIVISION, CSD
PRACTICAL EXERCISE NUMBER 2
PROGRAM NAME: DUMP-PF
VALUE IS +.2.
VALUE IS +.15.
VALUE IS +.12.
VALUE IS +.09.
VALUE IS +.05.
VALUE IS +.0.
VALUE IS 0.
VALUE IS 1.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
PROCEDURE DIVISION.
OPEN INPUT PERSONNEL-MASTER-FILE.
MOVE WS-SWITCH-OFF TO WS-MASTER-FILE-EOF-SWITCH.
PERFORM 0010-READ-AND-PRINT THRU 0010-EXIT UNTIL WS-MASTER-FILE-E0F-SWITCH EQUAL TO WS-SWITCH-ON.
CLOSE PERSONNEL-MASTER-FILE.
STOP RUN.
READ PERSONNEL-MASTER-FILE AT END.
PERFORM 0020-TOTALS-RTN THRU 0020-EXIT.
MOVE WS-SWITCH-ON TO WS-MASTER-FILE-E0F-SWITCH.
GO TO 0010-EXIT.
IF WS-DEPARTMENT-HOLD NOT EQUAL TO IM01-DEPARTMENT.
PERFORM 0020-TOTALS-RTN THRU 0020-EXIT.
PERFORM 0030-HEADING-RTN THRU 0030-EXIT.
IF IM01-NUMBER-DEPENDENTS EQUAL TO ZEROS.
MOVE WS-0-DEPENDENTS-TAX-RATE TO WS-TAX-RATE-HOLD.
ELSE.
IF IM01-NUMBER-DEPENDENTS LESS THAN 5.
MOVE WS-1-4-DEPENDENTS-TAX-RATE TO WS-TAX-RATE-HOLD.
ELSE.
IF IM01-NUMBER-DEPENDENTS LESS THAN 9.
VALUE IS SPACES.
PIC $99.99.
PIC X(15).
VALUE IS SPACES.
PIC VZ9.
PIC X(17).
VALUE IS %.
 PIC $99.99.
 PIC X(13).
 VALUE IS SPACES.
 PIC $99.99.
 PIC X(11).
 VALUE IS SPACES.
 PIC X(35).
 VALUE IS SPACES.
 PIC X(13).
 VALUE IS SPACES.
 PIC X(3).
 VALUE IS SPACES.
 PIC 9.
 PIC X(111).
 VALUE IS SPACES.
 PIC 9.
 PIC X(7).
 VALUE IS SPACES.
 PIC 9.
 PIC X(3).
 VALUE IS SPACES.
 PIC 9.
 PIC X(11).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
 PIC 9.
 PIC X(1).
 VALUE IS SPACES.
MOVING WS-5-8-DEPENDENTS TAX-RATE TO WS-TAX-RATE-HOLD
ELSE
IF IMCI-NUMBER-DEPENDENTS LESS THAN 13
MOVE WS-5-12-DEPENDENTS-TAX-RATE TO WS-TAX-RATE-HOLD
ELSE
MOVING WS-13-55-DEPENDENTS-TAX-RATE TO WS-TAX-RATE-HOLD.
MULTIPLY IMCI-MONTHLY-PAY BY WS-TAX-RATE-HOLD
GIVING WS-FEDERAL-TAX-HOLD.
MULTIPLY IMCI-MONTHLY-PAY BY WS-FICA-TAX-RATE
GIVING WS-FICA-TAX-HOLD.
SUBTRACT WS-FICA-TAX-HOLD WS-FEDERAL-TAX-HOLD FROM
IMCI-MONTHLY-PAY
GIVING WS-Net-Pay-Hold.
DIVIDE WS-Net-Pay-Hold BY WS-FAMILY-SIZE-HOLD
MOVE IMCI-NAME TO WS-NAME.
MOVING IMCI-MONTHLY-PAY TO IMCI-MONTHLY-PAY.
MOVE IMCI-NUMBER-DEPENDENTS TO WS-NUMBER-DEPENDENTS.
MOVE WS-FEDERAL-TAX-HOLD TO WS-FEDERAL-TAX.
MOVE WS-TAX-RATE-HOLD TO WS-TAX-RATE.
MOVE WS-FICA-TAX-HOLD TO WS-FICA-TAX.
MOVE WS-Net-Pay-Hold TO WS-Net-Pay.
ADD WS-Net-Pay-Hold TO WS-TOTAL-Net-Pay-SUM.
ADD WS-FAMILY-SIZE-HOLD TO WS-TOTAL-FAMILY-SIZE-SUM.
WRITE O001-Print-RCO FROM WS-DETAI-LINE
AFTER ADVANCING 2 LINES.
CC186C 0010-EXIT. EXIT.
CC137C CC02-TOALS-RTN.
MOVING WS-TOTAL-FAMILY-SIZE-SUM TO WS-TOTAL-FAMILY-SIZE.
MOVING WS-TOTAL-Net-Pay-SUM TO WS-TOTAL-Net-Pay.
WRITE O001-Printer-RCO FROM WS-FCOTING-LINE
AFTER ADVANCING 3 LINES.
MOVE ZEPCS TO WS-TOTAL-Net-Pay-SUM.
MOVING WS-TOTAL-Net-Pay-SUM TO WS-TOTAL-NET-PAY-SUM.
MOVE O001-DEPARTMENT TO WS-DEPARTMENT-HOLD.
WRITE O001-Printer-RCO FROM WS-COLUMN-HEADING
AFTER ADVANCING C H A N L.
00190 00190 0030-EXIT. EXIT.
00196 00196 0030-HEADING-RT.
00197 00197 0030-HEADING-RT.
00198 00198 0030-HEADING-RT.
00199 00199 0030-HEADING-RT.
00200 00200 0030-EXIT. EXIT.

IA-02-03-15

275
<table>
<thead>
<tr>
<th>INTERNAL NAME</th>
<th>LVL SOURCE NAME</th>
<th>BASE</th>
<th>DISPL</th>
<th>INTERNAL NAME</th>
<th>DEFINITION</th>
<th>USAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DN1=1-151</td>
<td>FD PERSONNEL-MASTER-FILE</td>
<td>DTF=01</td>
<td>BL=1</td>
<td>DN1=1-151</td>
<td>DS OCL30</td>
<td>DTF5D</td>
</tr>
<tr>
<td>DN1=1-196</td>
<td>01 1MC1-PERSONNEL-WASTE-RCD</td>
<td></td>
<td>JJO</td>
<td>DN1=1-196</td>
<td>DS 9C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=1-234</td>
<td>02 1MO1-SEC-SEC-AMU</td>
<td></td>
<td>JJO</td>
<td>DN1=1-234</td>
<td>DS 1B</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=1-260</td>
<td>02 1MO1-NAME</td>
<td></td>
<td>JJO</td>
<td>DN1=1-260</td>
<td>DS 2C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=1-327</td>
<td>02 1MO1-DEPARTMENT</td>
<td></td>
<td>JJO</td>
<td>DN1=1-260</td>
<td>DS 5C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=1-334</td>
<td>02 1MC1-NUMBER-DEPENDENTS</td>
<td></td>
<td>JJO</td>
<td>DN1=1-320</td>
<td>DS 2C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=1-352</td>
<td>02 1MO1-MONTHLY-PAY</td>
<td></td>
<td>JJO</td>
<td>DN1=1-352</td>
<td>DS 6C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=1-379</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=1-378</td>
<td>DS 38C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=1-417</td>
<td>02 PRINTER</td>
<td>DTF=02</td>
<td>BL=2</td>
<td>DN1=1-423</td>
<td>DS 133C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=1-428</td>
<td>01 CRC1-PRINTER-RC2</td>
<td></td>
<td>JJO</td>
<td>DN1=1-424</td>
<td>DS 1C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=1-454</td>
<td>77 WS-MASTER-FILE-ECF-SWİTCH</td>
<td></td>
<td>JJO</td>
<td>DN1=2-300</td>
<td>DS 2P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-300</td>
<td>77 WS-TOTAL-DEPARTMENT-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-334</td>
<td>DS 4P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-034</td>
<td>77 WS-EXECUTIVE-DEPARTMENT-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-367</td>
<td>DS 4P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-124</td>
<td>77 WS-MICROCOMPUTER-MAINTENANCE-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-124</td>
<td>DS 4P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-125</td>
<td>77 WS-DEPARTMENT-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-150</td>
<td>DS 4P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-150</td>
<td>77 WS-EXECUTIVE-DEPARTMENT-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-178</td>
<td>DS 2P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-207</td>
<td>77 WS-FAMILY-MAINTENANCE-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-207</td>
<td>DS 2P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-233</td>
<td>77 WS-FAMILY-MAINTENANCE-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-233</td>
<td>DS 2P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-256</td>
<td>77 WS-FAMILY-MAINTENANCE-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-259</td>
<td>DS 2P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-293</td>
<td>77 WS-EXECUTIVE-DEPARTMENT-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-293</td>
<td>DS 2P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-329</td>
<td>77 WS-EXECUTIVE-DEPARTMENT-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-329</td>
<td>DS 2P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-365</td>
<td>77 WS-EXECUTIVE-DEPARTMENT-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-365</td>
<td>DS 2P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-402</td>
<td>77 WS-EXECUTIVE-DEPARTMENT-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-402</td>
<td>DS 2P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-440</td>
<td>77 WS-EXECUTIVE-DEPARTMENT-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-440</td>
<td>DS 2P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DN1=2-462</td>
<td>77 WS-EXECUTIVE-DEPARTMENT-HOLD</td>
<td></td>
<td>JJO</td>
<td>DN1=2-462</td>
<td>DS 2P</td>
<td>COMP-3</td>
</tr>
<tr>
<td>DTN=3-C001</td>
<td>01 WS-COLUMN-HEADIN</td>
<td></td>
<td>JJO</td>
<td>DN1=3-003</td>
<td>DS 14C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-030</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-030</td>
<td>DS 14C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-049</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-049</td>
<td>DS 18C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-068</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-068</td>
<td>DS 15C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-087</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-087</td>
<td>DS 13C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-106</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-106</td>
<td>DS 11C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-125</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-125</td>
<td>DS 10C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-144</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-144</td>
<td>DS 11C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-163</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-163</td>
<td>DS 11C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-182</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-182</td>
<td>DS 11C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-201</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-201</td>
<td>DS 11C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-231</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-231</td>
<td>DS 11C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-250</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-250</td>
<td>DS 9C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-274</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-274</td>
<td>DS 5C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-293</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-293</td>
<td>DS 18C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-310</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-310</td>
<td>DS 6C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-329</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-329</td>
<td>DS 6C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-364</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-364</td>
<td>DS 6C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-402</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-402</td>
<td>DS 6C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-420</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-420</td>
<td>DS 6C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-439</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-439</td>
<td>DS 6C</td>
<td>DISP</td>
</tr>
<tr>
<td>DN1=3-474</td>
<td>02 FILLER</td>
<td></td>
<td>JJO</td>
<td>DN1=3-474</td>
<td>DS 6C</td>
<td>DISP</td>
</tr>
</tbody>
</table>
# MEMORY MAP

<table>
<thead>
<tr>
<th>TGT</th>
<th>03E4B</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAVE AREA</td>
<td>C054B</td>
</tr>
<tr>
<td>SWITCH</td>
<td>C0550</td>
</tr>
<tr>
<td>TALLY</td>
<td>C0554</td>
</tr>
<tr>
<td>SOFT SAVE</td>
<td>C0558</td>
</tr>
<tr>
<td>ENTRY-SAVE</td>
<td>C055C</td>
</tr>
<tr>
<td>SORT CORE SIZE</td>
<td>C0560</td>
</tr>
<tr>
<td>NSTC-REELS</td>
<td>C0564</td>
</tr>
<tr>
<td>SCFT SET</td>
<td>C0566</td>
</tr>
<tr>
<td>WORKING CELLS</td>
<td>C0568</td>
</tr>
<tr>
<td>SORT FILE SIZE</td>
<td>C0668</td>
</tr>
<tr>
<td>SORT MODE SIZE</td>
<td>C066C</td>
</tr>
<tr>
<td>PGT-VN TBL</td>
<td>C0670</td>
</tr>
<tr>
<td>TGT-VN TBL</td>
<td>C0674</td>
</tr>
<tr>
<td>SORTAB ADDRESS</td>
<td>C0678</td>
</tr>
<tr>
<td>LENGTH OF VN TBL</td>
<td>C067C</td>
</tr>
<tr>
<td>LENGTH OF SORTAB</td>
<td>C0680</td>
</tr>
<tr>
<td>PGM ID</td>
<td>C0684</td>
</tr>
<tr>
<td>A(INIT1)</td>
<td>C0688</td>
</tr>
<tr>
<td>UPSI SWITCHES</td>
<td>C068C</td>
</tr>
<tr>
<td>CVERFLOW CELLS</td>
<td>C0704</td>
</tr>
<tr>
<td>BL CELLS</td>
<td>C0708</td>
</tr>
<tr>
<td>CTFACR CELLS</td>
<td>C0710</td>
</tr>
<tr>
<td>TEMP STORAGE</td>
<td>C0718</td>
</tr>
<tr>
<td>TEMP STORAGE-2</td>
<td>C0720</td>
</tr>
<tr>
<td>TEMP STORAGE-3</td>
<td>C0728</td>
</tr>
<tr>
<td>TEMP STORAGE-4</td>
<td>C0730</td>
</tr>
<tr>
<td>BLL CELLS</td>
<td>C0738</td>
</tr>
<tr>
<td>VLC CELLS</td>
<td>C073C</td>
</tr>
<tr>
<td>SPL CELLS</td>
<td>C073C</td>
</tr>
<tr>
<td>INDEX CELLS</td>
<td>C073C</td>
</tr>
<tr>
<td>SUPAOR CELLS</td>
<td>C073C</td>
</tr>
<tr>
<td>CNCTL CELLS</td>
<td>C073C</td>
</tr>
<tr>
<td>PFMTL CELLS</td>
<td>C073C</td>
</tr>
<tr>
<td>PFMSAV CELLS</td>
<td>C073C</td>
</tr>
<tr>
<td>VN CELLS</td>
<td>C073C</td>
</tr>
<tr>
<td>SAVE AREA =2</td>
<td>C0750</td>
</tr>
<tr>
<td>XSASW CELLS</td>
<td>C075C</td>
</tr>
<tr>
<td>XSA CELLS</td>
<td>C075C</td>
</tr>
<tr>
<td>PACAM CELLS</td>
<td>C075C</td>
</tr>
<tr>
<td>RPTSAV AREA</td>
<td>C075C</td>
</tr>
<tr>
<td>CHECKPT CTR</td>
<td>C075C</td>
</tr>
<tr>
<td>IOPTR CELLS</td>
<td>C075C</td>
</tr>
</tbody>
</table>

---

## GENERAL POOL (HEX)

<table>
<thead>
<tr>
<th>LOC (LIT=0)</th>
<th>C05C0C0C1</th>
<th>0C4G2059</th>
<th>202212120</th>
<th>4B0220420</th>
<th>202120FC</th>
<th>20212000</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC (LIT=24)</td>
<td>0C0000C0</td>
<td>0C00C004</td>
<td>5B5C2C2C</td>
<td>07C9D540</td>
<td>5B5C2C2C</td>
<td>D3D6E2C5</td>
</tr>
</tbody>
</table>

IA-02-03-15

277
REGISTER ASSIGNMENT

REG 6  PL = 3
REG 7  PL = 1
REG 8  PL = 2

132

START

EOJ  *

L  1,1C3(0,13)  DTF=1
SH  1,086(0,12)  LIT+30
NL  000(1),X^5F
LA  1,086(0,12)  LIT+32
L  0,1C8(0,13)  DTF=1
LR  4,0
BALR  15,0
ST  0,009(0,15)
BAL  0,00C(0,15)
DC  X$0000000*
SVC  2
LA  1,086(0,12)  LIT+32
L  0,1C8(0,13)  DTF=2
LR  4,0
9CR  0,0
BALR  15,0
ST  0,006(0,15)
BAL  0,00C(0,15)
DC  X$0000000*
SVC  2

134

135

IA-02-03-'5
<table>
<thead>
<tr>
<th>PHASE</th>
<th>XFR-AD</th>
<th>SECORE</th>
<th>CSK-AC</th>
<th>ESD TYPE</th>
<th>LABEL</th>
<th>LOADED</th>
<th>REL-PR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHASE*** 005000 005000 006213 2E 12 4</td>
<td>CSECT</td>
<td>DUMPOE</td>
<td>005000 005000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CSECT</td>
<td>IJGF1IZ</td>
<td>005E28 005E28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENTRY</td>
<td>IJGF1IZ</td>
<td>005E28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENTRY</td>
<td>IJGF2IZ</td>
<td>005E28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENTRY</td>
<td>IJGF3IZ</td>
<td>035E28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CSECT</td>
<td>ILBDS0E</td>
<td>006000 006D00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENTRY</td>
<td>ILBDS1E</td>
<td>006D00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CSECT</td>
<td>IJDFA1IZ</td>
<td>005D10 005D10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENTRY</td>
<td>IJDFA2IZ</td>
<td>005D10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CSECT</td>
<td>ILBDS0S</td>
<td>0060CB 0060CB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WTRN</td>
<td>TRN</td>
<td>STXIT2SW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WTRN</td>
<td>TRN</td>
<td>ILBD00G2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0061C</td>
<td>0005110 4014478C FC0510C 4015471C</td>
<td>000512C 40024780 FO024130 00104700</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>0061D</td>
<td>0035811 00344154 00044550 F11455E5</td>
<td>CCCC4155 0001255 478F04C 18350D01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0061E</td>
<td>FCE43CCC 47F0FCSA 41330002 47F0F049</td>
<td>505FOEFA 05033006 F0E34730 F0E3E403</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0061F</td>
<td>00064432 CCA1E92 92FC5000 92CE53C1</td>
<td>5CC123C 478F09A 490DF0E6 478F092</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00620</td>
<td>56015602 47F0F30E 06015401 47F0F049</td>
<td>463155EE 59EFOEEE 58350D02 48F0F0EA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00621</td>
<td>27FF35C0 01845940 F11C1604 4110110A</td>
<td>C2C5109E 0255478C F0035810 F1125F00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00622</td>
<td>F1160701 1C21C21 4110033C 55D0103C</td>
<td>57F1F300 0A060000 00013630 00113039</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00623</td>
<td>47C91043 15CD4770 F19442EB 1C3985E5</td>
<td>CCE4690 F19C41CC 000142CB 103B0503</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00624</td>
<td>5839C2C7 96C2C509 33C00300 20000000</td>
<td>C04D0038 00000000 00033030 00000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00625</td>
<td>C0C0C0C0 --SAME--</td>
<td>C0C0C0C0 0C0C0C00 C0C0C0C0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 00626 | C0C0C0C0 CG0C0C0G 0C0C0C00 C0C0C0C0 |

**PAGE 14**

---

IA-02-03-15
PROGRAM NAME: DUMP-PE
**IRy cs AMERICAN NATIONAL STANDARD COBOL**

00001
00002
00003
00004
00005
00006
00007
00008
00009
00010
00011
00012
00013
00014
00015
00016
00017
00018
00019
00020
00021
00022
00023
00024
00025
00026
00027
00028
00029
00030
00031
00032
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052

**CRL LIP**

00001 BASIS C32DMP
00002 CC001C IDENTIFICATION DIVISION.
00003 CCC02C PROGRAM-ID. DUMP-PF.
00004 CCC03C AUTHOF. LT HUDDIN.
00005 CCC04C REMARKS. PE DESIGNED FOR USING ALC TO DEBUG A COROL PROGRAM.
00006 CCC05C ENVIRONMENT DIVISION.
00007 CO006C CONFIGURATION SECTION.
00008 CO007C SPECIAL-NAMES.
00009 CC008C CO1 IS CHAIN.
00010 CC009C INPUT-OUTPUT SECTION.
00011 CC010C FILE-CONTROL.
00012 CC011C SELECT PERSONNEL-MASTER-FILE
00013 CO012C ASSIGN TO SYS06-UT-2314-S.
00014 CC013C SELECT PRINT.
00015 CO014C ASSIGN TO SYS05-HR-1403-S.
00016 CC015C DATA DIVISION.
00017 CO016C FILE SECTION.
00018 CC017C FD PERSONNEL-MASTER-FILE
00019 CO018C LABEL RECORDS STANCARC.
00020 CC019C 01 IM01-PERSONNEL-MASTER-RC4.
00021 CC020C 05 IMC1-INDEX-SEC-INDEX
00022 CO021C 10 IMC1-NAME
00023 CC022C 05 IMC1-DEPARTMENT
00024 CO023C 05 FILLER
00025 CO024C 05 IMC1-NUMBER-DEPENDENTS
00026 CO025C 05 IM01-MCATHLY-PAY
00027 CO026C 05 FILLER
00028 CC027C FD PRINTER
00029 CC028C LABEL RECORDS OMITTED.
00030 CC029C 01 GR01-PRINTER-RC4
00031 CC030C WORKING-STORAGE SECTION.
00032 CO031C 77 WS-MASTER-FILE-EOF-SWITCH
00033 CC032C 77 WS-TOTAL-FAMILY-SIZE-SUM
00034 CC033C 01 IMC1-INDEX-SEC-INDEX
00035 CO034C 77 WS-TOTAL-NET-PAY-SUM
00036 CO035C 77 WS-DEPARTMENT-HOLC
00037 CO036C 77 WS-FEDERAL-TAX-HOLC
00038 CO037C 77 WS-FICA-TAX-HOLC
00039 CO038C 77 WS-NFT-PAY-HOLC
00040 CO039C 77 WS-FAMILY-SIZE-HOLC
00041 CO040C 77 WS-FAMILY-SIZE-HOLC
00042 CO041C 77 WS-FAMILY-SIZE-HOLC
00043 CO042C 77 WS-FAMILY-SIZE-HOLC
00044 CO043C 77 WS-FAMILY-SIZE-HOLC
00045 CO044C 77 WS-FAMILY-SIZE-HOLC
00046 CO045C 77 WS-FAMILY-SIZE-HOLC
00047 CO046C 77 WS-FAMILY-SIZE-HOLC
00048 CO047C 77 WS-FAMILY-SIZE-HOLC
00049 CO048C 77 WS-FAMILY-SIZE-HOLC
00050 CC049C 77 WS-FAMILY-SIZE-HOLC
00051 CC050C 77 WS-O-DEPENDENTS-TAX-RATE
00052 CC051C 77 WS-O-DEPENDENTS-TAX-RATE
00053 000510
00054 00052C
00055 000530 77  WS-1-4-DEPENDENTS-TAX-RATE  USAGE IS COMP-3  VALUE IS +.2.
00056 000540
00057 00055C
00058 000560 77  WS-5-8-DEPENDENTS-TAX-RATE  USAGE IS COMP-3  VALUE IS +.15.
00059 00057C
00060 00058C
00061 00059C 77  WS-9-12-DEPENDENTS-TAX-RATE  USAGE IS COMP-3  VALUE IS +.12.
00062 00060C
00063 00061C
00064 000620 77  WS-13-59-DEPENDENTS-TAX-RATE  USAGE IS COMP-3  VALUE IS +.08.
00065 000630
00066 00064C
00067 000650 77  WS-SWITCH-CN  USAGE IS COMP-3  VALUE IS +.05.
00068 00066C
00069 000670 77  WS-SWITCH-OFF  USAGE IS COMP-3  VALUE IS '0'.
00070 00068C
00071 00069C 01  WS-COLUMN-HEADING.
00072 00070C 05 FILLER
00073 00071C
00074 000720 05 FILLER
00075 00073C
00076 00074C 05 FILLER
00077 00075C
00078 00076C 05 FILLER
00079 00077G
00080 00078C C5 FILLER
00081 00079G
00082 00080C 05 FILLER
00083 00081G
00084 00082C C5 FILLER
00085 00083C
00086 00084C C5 FILLER
00087 00085C
00088 00086C 05 FILLER
00089 00087C
00090 CCCPR4 01  WS-DETAIL-LINE.
00091 00090C C5 FILLER
00092 000910
00093 000920 05 WS-SSC-SEC-NUM  VALUE SPACES.
00094 000930
00095 00094C
00096 00095C C5 WS-NAME
00097 00096C
00098 00097C
00099 00098C 05 WS-PMTLY-PAY  VALUE SPACES.
01000 00099C
01001 00100C
01002 00101C C5 WS-NUMBER-DEPENDENTS  VALUE SPACES.
01003 010020 05 FILLER
01004 00103C
01005 00104C C5 WS-PAY-AVERAGE  VALUE SPACES.

USAGE IS COMP-3
PIC SV9(3)
LSAGE IS COMP-3
PIC SV9(3)
USAGE IS COMP-3
PIC SV9(3)
USAGE IS COMP-3
PIC SV9(3)
USAGE IS COMP-3
PIC SV9(3)
USAGE IS COMP-3
PIC SV9(3)
USAGE IS COMP-3
PIC SV9(3)
USAGE IS COMP-3
PIC SV9(3)
VALUE IS '0'.
VALUE IS '1'.
VALUE IS 'MONTHLY PAY'.
VALUE IS 'NO OF DEPS'.
VALUE IS 'PAY AVE'.
VALUE IS 'FED TAX'.
VALUE IS 'FICA TAX'.
VALUE IS 'NET-PAY'.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS 'NAME'.
VALUE IS 'SSN'.
VALUE IS '999.99.'
VALUE IS '999.99.'
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
VALUE IS SPACES.
PROCEDURE DIVISION.
OPEN INPUT PERSONNEL-MASTER-FILE
OUTPUT PRINTER.
MOVE WS-SWITCH-OFF TO WS-MASTER-FILE-EOF-SWITCH.
PERFORM 0010-READ-AND-PRINT THRU 0010-EXIT UNTIL WS-MASTER-FILE-EOF-SWITCH EQUAL TO WS-SWITCH-ON.
CLOSE PERSONNEL-MASTER-FILE PRINTER.
STOP RUN.
0010 READ PERSONNEL-MASTER-FILE
AT END
PERFORM 0020-TOTALS-RTN THRU 0020-EXIT
MOVE WS-SWITCH-CK TO WS-MASTER-FILE-END-SWITCH
GO TO 0010-EXIT.
IF WS-DEPARTMENT-HELIC EQUAL TO SPACES
PERFORM 0030-READING-RTN THRU 0030-EXIT
ELSIF IF WS-DEPARTMENT-HELIC NOT EQUAL TO IM01-DEPARTMENT
PERFORM 0040-TO-0 TOTALS-RTN THRU 0040-EXIT
PERFORM 0030-READING-RTN THRU 0030-EXIT.
IF IM01-NUMBER-DEPENDENTS EQUAL TO ZEROS
MOVE WS-3-DEPENDENTS-TAX-RATE TO WS-TAX-RATE-HOLD
ELSIF IF IM01-NUMBER-DEPENDENTS LESS THAN 5
MOVE WS-1-4-DEPENDENTS-TAX-RATE TO WS-TAX-RATE-HOLD
ELSE
TC IMO1-NUMBER-DEPENDENTS LESS THAN 4
MOVE WS-5-9-DEPENDENTS-TAX-RATE TO WS-TAX-RATE-HOLD
ELSE
ENDIF
MOVE WS-9-12-DEPENDENTS-TAX-RATE TO WS-TAX-RATE-HOLD
ELSE
MOVE WS-13-99-DEPENDENTS-TAX-RATE TO
WS-TAX-RATE-HOLD.
MULTIPLY IM01-MONTHLY-PAY BY WS-TAX-RATE-HOLD
GIVING WS-FEDERAL-TAX-HOLD.
MULTIPLY IM01-MONTHLY-PAY BY WS-FICA-TAX-RATE
GIVING WS-FICA-TAX-HOLD.
SUBTRACT WS-FICA-TAX-HOLD WS-FEDERAL-TAX-HOLD FROM
IM01-MONTHLY-PAY
GIVING WS-NET-PAY-HOLD.
DIVIDE WS-NET-PAY-HOLD BY WS-FAMILY-SIZE-HOLD
GIVING WS-PAY-AVERAGE.
MOVE IM01-SCC-SEC-AMP TO WS-SCC-SEC-NUM.
MOVE IM01-NAME TO WS-NAME.
MOVE IM01-MONTHLY-PAY TO WS-MONTHLY-PAY.
MOVE IM01-NUMBER-DEPENDENTS TO WS-NUMBER-DEPENDENTS.
MOVE WS-FEDERAL-TAX-HOLD TO WS-FEDERAL-TAX.
MOVE WS-TAX-RATE-HOLD TO WS-TAX-RATE.
MOVE WS-FICA-TAX-HOLD TO WS-FICA-TAX.
MOVE WS-NET-PAY-HOLD TO WS-NET-PAY.
ADD WS-NET-PAY-HOLD TO WS-TOTAL-NET-PAY-SUM.
ADD WS-FAMILY-SIZE-HOLD TO WS-TOTAL-FAMILY-SIZE-SUM.
WRITE OR01-PRINTER-RCDF FROM WS-DETAIL-LINE
AFTER ADVANCING 2 LINES.
MOVE ZEROS TO WS-TOTAL-FAMILY-SIZE-SUM.
MOVE ZEROS TO WS-TOTAL-NET-PAY-SUM.
WRITE OR01-PRINTER-RCDF FROM WS-FOOTING-LINE
AFTER ADVANCING 3 LINES.
MOVE ZEROS TO WS-TOTAL-FAMILY-SIZE-SUM.
MOVE ZEROS TO WS-TOTAL-NET-PAY-SUM.
WRITE OR01-PRINTER-RCDF FROM WS-COLUMN-HEADING
AFTER ADVANCING COLUMNS.
END.

<table>
<thead>
<tr>
<th>INTRNL NAME</th>
<th>LVL SOURCE NAME</th>
<th>BASE DISPL</th>
<th>INTRNL NAME</th>
<th>DEFINITION</th>
<th>USAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNW=1-151</td>
<td>FC PERSONNEL-MASTER-FILE</td>
<td>DTF=01</td>
<td>DNW=1-151</td>
<td>DTF50</td>
<td>F</td>
</tr>
<tr>
<td>DNW=1-154</td>
<td>01 IMO1-FFRPERSONNEL-MASTER-PCD</td>
<td>BL=1 000</td>
<td>DNW=1-196</td>
<td>DS OCLBO GROUP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-234</td>
<td>02 IMO1-SOC-SEC-NUM</td>
<td>BL=1 000</td>
<td>DNW=1-234</td>
<td>DS 9C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-260</td>
<td>02 IMO1-NAME</td>
<td>BL=1 009</td>
<td>DNW=1-260</td>
<td>DS 18C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-279</td>
<td>02 IMO1-DEPARTMENT</td>
<td>AL=1 018</td>
<td>DNW=1-279</td>
<td>DS 2C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-304</td>
<td>02 FILLER</td>
<td>BL=1 010</td>
<td>DNW=1-304</td>
<td>DS 5C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-320</td>
<td>02 IMO1-MEMBER-DEPENDENTS</td>
<td>BL=1 022</td>
<td>DNW=1-320</td>
<td>DS 2C DISP-NM</td>
<td></td>
</tr>
<tr>
<td>DNW=1-352</td>
<td>02 IMO1-RIGHTLY-PAY</td>
<td>BL=1 024</td>
<td>DNW=1-352</td>
<td>DS 6C DISP-NM</td>
<td></td>
</tr>
<tr>
<td>DNW=1-378</td>
<td>02 FILLER</td>
<td>BL=1 024</td>
<td>DNW=1-378</td>
<td>DS 38C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-457</td>
<td>FD PRINTER</td>
<td></td>
<td>DNW=1-397</td>
<td>DTFPR</td>
<td>F</td>
</tr>
<tr>
<td>DNW=1-488</td>
<td>01 ORC1-PRINTER-RCC</td>
<td>BL=2 000</td>
<td>DNW=1-428</td>
<td>DS 13C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-489</td>
<td>77 WS-MASTER-FILE-ECF-SWITCH</td>
<td>BL=3 000</td>
<td>DNW=1-454</td>
<td>DS 1C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-540</td>
<td>77 WS-TOTAL-FAMILY-SIZE-SUM</td>
<td>BL=3 021</td>
<td>DNW=2-030</td>
<td>DS 2P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-587</td>
<td>77 WS-TOTAL-NET-PAY-SUM</td>
<td>BL=3 003</td>
<td>DNW=2-034</td>
<td>DS 4P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-674</td>
<td>77 WS-DEPARTMENT-CLC</td>
<td>BL=3 007</td>
<td>DNW=2-067</td>
<td>DS 2G DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-667</td>
<td>77 WS-FEEER-AL-TAX-HOL</td>
<td>BL=3 029</td>
<td>DNW=2-095</td>
<td>DS 4P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-124</td>
<td>77 WS-FICA-TAX-CLD</td>
<td>BL=3 000</td>
<td>DNW=2-124</td>
<td>DS 4P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-150</td>
<td>77 WS-NET-PAY-HOLD</td>
<td>BL=3 011</td>
<td>DNW=2-153</td>
<td>DS 4P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-178</td>
<td>77 WS-FAMILY-SIZE-HOLD</td>
<td>BL=3 015</td>
<td>DNW=2-178</td>
<td>DS 2P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-270</td>
<td>77 WS-TAX-RATE-CLC</td>
<td>BL=3 017</td>
<td>DNW=2-207</td>
<td>DS 2P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-293</td>
<td>77 WS-FICA-TAX-RATE</td>
<td>BL=3 019</td>
<td>DNW=2-233</td>
<td>DS 2P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-294</td>
<td>77 WS-0-CEPENT-CENTS-TAX-RATE</td>
<td>BL=3 019</td>
<td>DNW=2-259</td>
<td>DS 2P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-295</td>
<td>77 WS-1-4-CEPENT-CENTS-TAX-RATE</td>
<td>BL=3 010</td>
<td>DNW=2-293</td>
<td>DS 2P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-329</td>
<td>77 WS-5-8-CEPENT-CENTS-TAX-RATE</td>
<td>BL=3 01F</td>
<td>DNW=2-329</td>
<td>DS 2P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-365</td>
<td>77 WS-9-12-CEPENT-CENTS-TAX-RATE</td>
<td>BL=3 021</td>
<td>DNW=2-365</td>
<td>DS 2P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-430</td>
<td>77 WS-13-99-CEPENT-CENTS-TAX-RATE</td>
<td>BL=3 023</td>
<td>DNW=2-402</td>
<td>DS 2P COMP-3</td>
<td></td>
</tr>
<tr>
<td>DNW=1-440</td>
<td>77 WS-SWITCH-ON</td>
<td>BL=3 025</td>
<td>DNW=2-440</td>
<td>DS 1C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-462</td>
<td>77 WS-SWITCH-OFF</td>
<td>BL=3 026</td>
<td>DNW=2-462</td>
<td>DS 1C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-500</td>
<td>01 WS-COLUMN-READING</td>
<td>BL=3 029</td>
<td>DNW=3-000</td>
<td>DS OCL122 GROUP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-530</td>
<td>02 FILLER</td>
<td>BL=3 028</td>
<td>DNW=3-330</td>
<td>DS 14C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-545</td>
<td>02 FILLER</td>
<td>BL=3 036</td>
<td>DNW=3-049</td>
<td>DS 18C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-560</td>
<td>02 FILLER</td>
<td>BL=3 048</td>
<td>DNW=3-068</td>
<td>DS 19C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-666</td>
<td>02 FILLER</td>
<td>BL=3 057</td>
<td>DNW=3-087</td>
<td>DS 13C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-684</td>
<td>02 FILLER</td>
<td>BL=3 064</td>
<td>DNW=3-100</td>
<td>DS 11C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-700</td>
<td>02 FILLER</td>
<td>BL=3 06F</td>
<td>DNW=3-125</td>
<td>DS 10C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-734</td>
<td>02 FILLER</td>
<td>BL=3 079</td>
<td>DNW=3-144</td>
<td>DS 11C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-763</td>
<td>02 FILLER</td>
<td>BL=3 084</td>
<td>DNW=3-163</td>
<td>DS 11C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-782</td>
<td>02 FILLER</td>
<td>BL=3 08F</td>
<td>DNW=3-182</td>
<td>DS 19C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-801</td>
<td>01 WS-DETAIL-LINE</td>
<td>BL=3 048</td>
<td>DNW=3-201</td>
<td>DS OCL133 GROUP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-831</td>
<td>01 WS-DETAIL-LINE</td>
<td>BL=3 048</td>
<td>DNW=3-231</td>
<td>DS OCL133 GROUP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-850</td>
<td>02 WS-SOC-SEC-NUM</td>
<td>BL=3 0R3</td>
<td>DNW=3-250</td>
<td>DS 9C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-874</td>
<td>02 FILLER</td>
<td>BL=3 0AC</td>
<td>DNW=3-274</td>
<td>DS 5C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-914</td>
<td>02 WS-NAME</td>
<td>BL=3 0C1</td>
<td>DNW=3-300</td>
<td>DS 18C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-926</td>
<td>02 WSC-ACTLY-PAY</td>
<td>BL=3 009</td>
<td>DNW=3-329</td>
<td>DS 8C N-EDIT</td>
<td></td>
</tr>
<tr>
<td>DNW=1-934</td>
<td>02 WS-CNACTLY-PAY</td>
<td>BL=3 00F</td>
<td>DNW=3-346</td>
<td>DS 6C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-964</td>
<td>02 FILLER</td>
<td>BL=3 0E7</td>
<td>DNW=3-383</td>
<td>DS 2C N-EDIT</td>
<td></td>
</tr>
<tr>
<td>DNW=1-993</td>
<td>02 WS-NAME</td>
<td>BL=3 0EF</td>
<td>DNW=3-420</td>
<td>DS 6C DISP</td>
<td></td>
</tr>
<tr>
<td>DNW=1-345</td>
<td>02 WS-PAY-AVERAGE</td>
<td>BL=3 0EF</td>
<td>DNW=3-439</td>
<td>DS 8C N-EDIT</td>
<td></td>
</tr>
<tr>
<td>DNW=1-474</td>
<td>07 FILLER</td>
<td>BL=3 0F7</td>
<td>DNW=3-474</td>
<td>DS 2C DISP</td>
<td></td>
</tr>
</tbody>
</table>
### MEMORY MAP

<table>
<thead>
<tr>
<th>TGT</th>
<th>00548</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAVE AREA</td>
<td>00549</td>
</tr>
<tr>
<td>SWITCH</td>
<td>CC950</td>
</tr>
<tr>
<td>TALLY</td>
<td>0C954</td>
</tr>
<tr>
<td>SCRT SAVE</td>
<td>0C95A</td>
</tr>
<tr>
<td>ENTPY-SAVE</td>
<td>0055C</td>
</tr>
<tr>
<td>SOFT CORE SIZE</td>
<td>005A0</td>
</tr>
<tr>
<td>ASTC-REELS</td>
<td>005A4</td>
</tr>
<tr>
<td>SORT RET</td>
<td>CC95E</td>
</tr>
<tr>
<td>WORKING CELLS</td>
<td>005A9</td>
</tr>
<tr>
<td>SOFT FILE SIZE</td>
<td>CC4DB</td>
</tr>
<tr>
<td>SORT MODE SIZE</td>
<td>CC4DC</td>
</tr>
<tr>
<td>PGT-VN TRL</td>
<td>CC4ED</td>
</tr>
<tr>
<td>TGT-VN TRL</td>
<td>006E4</td>
</tr>
<tr>
<td>SORTAR ADDRESS</td>
<td>0C6FA</td>
</tr>
<tr>
<td>LENGTH OF VN TRL</td>
<td>0C6FC</td>
</tr>
<tr>
<td>LENGTH OF SCRTAS</td>
<td>0C6FE</td>
</tr>
<tr>
<td>PGN ID</td>
<td>0C6F0</td>
</tr>
<tr>
<td>A(CINIT)</td>
<td>0C6FE</td>
</tr>
<tr>
<td>UPSI SWITCHES</td>
<td>006FC</td>
</tr>
<tr>
<td>OVERFLOW CELLS</td>
<td>CC7C4</td>
</tr>
<tr>
<td>RL CELLS</td>
<td>CC7C4</td>
</tr>
<tr>
<td>OTRACR CELLS</td>
<td>CC710</td>
</tr>
<tr>
<td>TEMP STORAGE</td>
<td>00718</td>
</tr>
<tr>
<td>TEMP STORAGE-2</td>
<td>CC72A</td>
</tr>
<tr>
<td>TEMP STORAGE-3</td>
<td>CC738</td>
</tr>
<tr>
<td>TEMP STORAGE-4</td>
<td>CC73A</td>
</tr>
<tr>
<td>BLL CELLS</td>
<td>00738</td>
</tr>
<tr>
<td>VLC CELLS</td>
<td>CC72C</td>
</tr>
<tr>
<td>SRL CELLS</td>
<td>0072C</td>
</tr>
<tr>
<td>INDEX CELLS</td>
<td>0072C</td>
</tr>
<tr>
<td>SUBACR CELLS</td>
<td>CC73C</td>
</tr>
<tr>
<td>GNCTL CELLS</td>
<td>CC73C</td>
</tr>
<tr>
<td>PFPCTL CELLS</td>
<td>0073C</td>
</tr>
<tr>
<td>PFP*SAV CELLS</td>
<td>0073C</td>
</tr>
<tr>
<td>VN CELLS</td>
<td>CC750</td>
</tr>
<tr>
<td>SAVE AREA #2</td>
<td>0375C</td>
</tr>
<tr>
<td>XSASW CELLS</td>
<td>0075C</td>
</tr>
<tr>
<td>XSA CELLS</td>
<td>0075C</td>
</tr>
<tr>
<td>PARAM CELLS</td>
<td>0075C</td>
</tr>
<tr>
<td>PPF*SAV AREA</td>
<td>0075C</td>
</tr>
<tr>
<td>CHECKPT CTR</td>
<td>0075C</td>
</tr>
<tr>
<td>TMTPR CELLS</td>
<td>CC75C</td>
</tr>
</tbody>
</table>

### LITERAL POOL (HEX)

| 007DD (LIT+0) | 0F5C001 3C40205R 20201220 4R202040 202120FC 20212000 |
| 007EE (LIT+24) | CCG0000 00000004 5A59C2C6 07C5F540 5A59C2C3 D306E2C5 |

IA-02-02-22

294
<table>
<thead>
<tr>
<th>PGT</th>
<th>OC769</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVERFLOW CELLS</td>
<td>CC76A</td>
</tr>
<tr>
<td>VIRTUAL CELLS</td>
<td>076A</td>
</tr>
<tr>
<td>PROCEDURE NAME CELLS</td>
<td>CC76C</td>
</tr>
<tr>
<td>GENERATED NAME CELLS</td>
<td>CC76C</td>
</tr>
<tr>
<td>BUFFER ADDRESS CELLS</td>
<td>CC76C</td>
</tr>
<tr>
<td>VNI CELLS</td>
<td>0C7C4</td>
</tr>
<tr>
<td>LITERALS</td>
<td>CC7C0</td>
</tr>
<tr>
<td>DISPLAY LITERALS</td>
<td>C08C0</td>
</tr>
</tbody>
</table>

**REGISTER ASSIGNMENT**

| REG 6 | RL = 3 |
| REG 7 | RL = 1 |
| REG 8 | RL = 2 |

**133**

<table>
<thead>
<tr>
<th>OCOA00</th>
<th>START</th>
<th>EQU</th>
</tr>
</thead>
<tbody>
<tr>
<td>00800C</td>
<td>5A 1D 1Ca</td>
<td>L 1,1C(0,13)</td>
</tr>
<tr>
<td>03CC64</td>
<td>6A 10 C 1C6</td>
<td>SH 1,088(0,12)</td>
</tr>
<tr>
<td>00B809</td>
<td>94 FF 1 00D</td>
<td>NI 0000111, X'8F'</td>
</tr>
<tr>
<td>00CC8C</td>
<td>41 1D C 0B8</td>
<td>LA 1,088(0,12)</td>
</tr>
<tr>
<td>3G0A10</td>
<td>58 00 D 1Ca</td>
<td>L 0,1C9(0,13)</td>
</tr>
<tr>
<td>CCA14</td>
<td>1A 40</td>
<td>LR 4,0</td>
</tr>
<tr>
<td>008B18</td>
<td>C5 F0</td>
<td>RALR 15,0</td>
</tr>
<tr>
<td>008B18</td>
<td>50 00 F 0CB</td>
<td>ST 0,000(0,15)</td>
</tr>
<tr>
<td>00CA1C</td>
<td>45 00 F 0CC</td>
<td>RAL 0,00C(0,15)</td>
</tr>
<tr>
<td>00DB2C</td>
<td>0000000</td>
<td>DC X'00000000'</td>
</tr>
<tr>
<td>00C242</td>
<td>0A 02</td>
<td>SVC 2</td>
</tr>
<tr>
<td>00C262</td>
<td>41 1D C 0B8</td>
<td>LA 1,088(0,12)</td>
</tr>
<tr>
<td>JCC824</td>
<td>58 00 F 1CC</td>
<td>L 0,1C9(0,13)</td>
</tr>
<tr>
<td>008B2E</td>
<td>1A 40</td>
<td>L 4,0</td>
</tr>
<tr>
<td>008P00</td>
<td>07 00</td>
<td>RCR 1,0</td>
</tr>
<tr>
<td>3CCA32</td>
<td>65 F0</td>
<td>RALP 15,0</td>
</tr>
<tr>
<td>00CA34</td>
<td>50 00 F 0CC</td>
<td>ST 0,008(0,15)</td>
</tr>
<tr>
<td>JCC832</td>
<td>45 00 F 0OC</td>
<td>BAL 0,00C(0,15)</td>
</tr>
<tr>
<td>00C83C</td>
<td>CC00000</td>
<td>DC X'00000000'</td>
</tr>
<tr>
<td>00C840</td>
<td>0A 02</td>
<td>SVC 2</td>
</tr>
<tr>
<td>000B42</td>
<td>50 2D D 1CD</td>
<td>ST 2,1C0(0,13)</td>
</tr>
<tr>
<td>01C846</td>
<td>58 4D D 1CD</td>
<td>L 8,1C0(0,13)</td>
</tr>
<tr>
<td>00C846</td>
<td>02 00 6 0C0 A 026</td>
<td>MVC 000111, 026(6)</td>
</tr>
<tr>
<td>008P50</td>
<td>58 00 7Ca</td>
<td>L 0,208(0,13)</td>
</tr>
<tr>
<td>00C854</td>
<td>50 00 D 1F4</td>
<td>ST 0,1F4(0,13)</td>
</tr>
<tr>
<td>00C858</td>
<td>58 00 C 01C</td>
<td>L 0,01C(0,12)</td>
</tr>
<tr>
<td>00C85C</td>
<td>50 00 C 0C9</td>
<td>ST 0,208(0,13)</td>
</tr>
<tr>
<td>00C86C</td>
<td>CA=01</td>
<td>EQU</td>
</tr>
<tr>
<td>00C86C</td>
<td>5A 20 C 02D</td>
<td>L 2,070(0,12)</td>
</tr>
<tr>
<td>00DBF4</td>
<td>D5 30 6 000 A 025</td>
<td>CLC 000111, 025(6)</td>
</tr>
<tr>
<td>0000AF</td>
<td>07 2F</td>
<td>SCP 8,2</td>
</tr>
<tr>
<td>00C86C</td>
<td>58 10 C 034</td>
<td>L 1,004(0,12)</td>
</tr>
</tbody>
</table>
### DATA NAMES

<table>
<thead>
<tr>
<th>DATA NAME</th>
<th>DEFN</th>
<th>REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERSONNEL-MASTER-FILE</td>
<td>00012</td>
<td>00133</td>
</tr>
<tr>
<td>IMC1-SEC-SEC-NUM</td>
<td>0021</td>
<td>00133</td>
</tr>
<tr>
<td>IM01-NAME</td>
<td>0022</td>
<td>00134</td>
</tr>
<tr>
<td>IMC1-DEPARTMENT</td>
<td>0023</td>
<td>00139</td>
</tr>
<tr>
<td>IM01-NUMMFR-DEPENDENTS</td>
<td>0025</td>
<td>00153</td>
</tr>
<tr>
<td>WP01-MONTHLY-PAY</td>
<td>0026</td>
<td>00169</td>
</tr>
<tr>
<td>PRINTER</td>
<td>0014</td>
<td>00171</td>
</tr>
<tr>
<td>OR01-PRINTER-RCO</td>
<td>0030</td>
<td>00184</td>
</tr>
<tr>
<td>W5-MASTER-FILE-ECF-SWITCH</td>
<td>0032</td>
<td>00185</td>
</tr>
<tr>
<td>W5-TOTAL-FAMILY-SIZE-SUM</td>
<td>0033</td>
<td>00186</td>
</tr>
<tr>
<td>W5-TOTAL-NET-PAY-SUM</td>
<td>0035</td>
<td>00187</td>
</tr>
<tr>
<td>W5-DEPARTMENT-HOLD</td>
<td>0037</td>
<td>00188</td>
</tr>
<tr>
<td>W5-FEDERAL-TAX-HCLD</td>
<td>0039</td>
<td>00189</td>
</tr>
<tr>
<td>W5-FICA-TAX-HCLD</td>
<td>0041</td>
<td>00190</td>
</tr>
<tr>
<td>W5-NET-PAY-HCLD</td>
<td>0043</td>
<td>00191</td>
</tr>
<tr>
<td>W5-FAMILY-SIZE-HCLD</td>
<td>0045</td>
<td>00192</td>
</tr>
<tr>
<td>W5-TAX-RATE-HOLD</td>
<td>0047</td>
<td>00193</td>
</tr>
<tr>
<td>W5-FICA-TAX-RATE</td>
<td>0049</td>
<td>00194</td>
</tr>
<tr>
<td>W5-0-DEPENDENTS-TAX-RATE</td>
<td>0052</td>
<td>00195</td>
</tr>
<tr>
<td>W5-1-4-DEPENDENTS-TAX-RATE</td>
<td>0055</td>
<td>00196</td>
</tr>
<tr>
<td>W5-5-8-DEPENDENTS-TAX-RATE</td>
<td>0058</td>
<td>00197</td>
</tr>
<tr>
<td>W5-9-12-DEPENDENTS-TAX-RATE</td>
<td>0061</td>
<td>00198</td>
</tr>
<tr>
<td>W5-13-99-DEPENDENTS-TAX-RATE</td>
<td>0064</td>
<td>00199</td>
</tr>
<tr>
<td>W5-SWITCH-CN</td>
<td>0067</td>
<td>00199</td>
</tr>
<tr>
<td>W5-SWITCH-CFF</td>
<td>0069</td>
<td>00199</td>
</tr>
<tr>
<td>W5-COLUMN-HEADING</td>
<td>0071</td>
<td>00199</td>
</tr>
<tr>
<td>W5-DETAIL-LINE</td>
<td>0073</td>
<td>00199</td>
</tr>
<tr>
<td>W5-SEC-SEC-NUM</td>
<td>0075</td>
<td>00199</td>
</tr>
<tr>
<td>W5-NAME</td>
<td>0077</td>
<td>00199</td>
</tr>
<tr>
<td>W5-MONTHLY-PAY</td>
<td>0079</td>
<td>00199</td>
</tr>
<tr>
<td>W5-NUMBER-DEPENDENTS</td>
<td>0082</td>
<td>00199</td>
</tr>
<tr>
<td>W5-PAY-AVERAGE</td>
<td>0085</td>
<td>00199</td>
</tr>
<tr>
<td>W5-FEDERAL-TAX</td>
<td>0088</td>
<td>00199</td>
</tr>
<tr>
<td>W5-TAX-RATE</td>
<td>0091</td>
<td>00199</td>
</tr>
<tr>
<td>W5-FICA-TAX</td>
<td>0094</td>
<td>00199</td>
</tr>
<tr>
<td>W5-NET-PAY</td>
<td>0097</td>
<td>00199</td>
</tr>
<tr>
<td>W5-FOOTING-LINE</td>
<td>0100</td>
<td>00199</td>
</tr>
<tr>
<td>W5-TOTAL-FAMILY-SIZE</td>
<td>0102</td>
<td>00199</td>
</tr>
<tr>
<td>W5-TOTAL-NET-PAY</td>
<td>0105</td>
<td>00199</td>
</tr>
</tbody>
</table>

### PROCEDURE NAMES

<table>
<thead>
<tr>
<th>PROCEDURE NAME</th>
<th>DEFN</th>
<th>REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010-READ-AND-PRINT</td>
<td>00141</td>
<td>00136</td>
</tr>
<tr>
<td>0010-EXIT</td>
<td>00143</td>
<td>00176</td>
</tr>
<tr>
<td>0020-TOTALS-RTN</td>
<td>00149</td>
<td>00180</td>
</tr>
<tr>
<td>0020-EXIT</td>
<td>00151</td>
<td>00182</td>
</tr>
</tbody>
</table>
**PHASE*** | C05000 | 005000 | 006213 | 2E 12 4 |
---|---|---|---|---|
**CSECT** | **CUMPOPE** | 005000 | 005000 |

**CSECT** | **I1GF1EWZ** | 005E28 | 005E28 |
**ENTRY** | **I1GF1IZ7** | 005E28 |
**ENTRY** | **I1GF1ZZZ** | 005E28 |
**ENTRY** | **I1GF1EZZ** | 005E28 |

**CSECT** | **ILR0SAE0** | 006000 | 006000 |
**ENTRY** | **ILR0SAE1** | 006000 |

**CSECT** | **IJDFAPIZ** | 005D10 | 005D10 |
**ENTRY** | **IJDFAZIZ** | 005D10 |

**CSECT** | **LADMNSO** | 0060CB | 0060CB |

**WXTRN** | **STXITPSW** |
**WXTRN** | **ILADDBG2** |

**UNREFERENCED SYMBOLS**

**002 UNRESOLVED ADDRESS CONSTANTS**
<table>
<thead>
<tr>
<th>SSN</th>
<th>NAME</th>
<th>MONTHLY PAY</th>
<th>NO OF DEPS</th>
<th>PAY AVE</th>
<th>FED TAX</th>
<th>FICA TAX</th>
<th>NET-PAY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DEBUG
USAIA
SOFTWARE DIVISION, CSD
PRACTICAL EXERCISE NUMBER 4
PROGRAM NAME: DUMP-PE
COL L19
00001 BASIS C320MP*
00002 GC001C IDENTIFICATION DIVISION.*
00003 GC0020 PROGRAM-ID. EMPE-PE.
00004 GC002C ENVIRONMENT DIVISION.*
00005 GC002C CONFIGURATION SECTION.*
00006 GC002C SPECIAL-NAME.
00007 GC0020 CO1 IS CHAIN.
00008 GC0020 CO1 IS CHAIN.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
00009 GC0020 CMRNY CRITICAL.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCC53</td>
<td>OCC49C</td>
<td>OCC50C</td>
<td>77</td>
</tr>
<tr>
<td>OCC54</td>
<td>OCC510</td>
<td>OCC520</td>
<td></td>
</tr>
<tr>
<td>OCC55</td>
<td>OCC530</td>
<td>OCC540</td>
<td>77</td>
</tr>
<tr>
<td>OCC56</td>
<td>OCC550</td>
<td>OCC560</td>
<td></td>
</tr>
<tr>
<td>OCC57</td>
<td>OCC566C</td>
<td>OCC570</td>
<td>77</td>
</tr>
<tr>
<td>OCC58</td>
<td>OCC580</td>
<td>OCC590</td>
<td></td>
</tr>
<tr>
<td>OCC59</td>
<td>OCC596C</td>
<td>OCC600</td>
<td>77</td>
</tr>
<tr>
<td>OCC60</td>
<td>OCC610</td>
<td>OCC620</td>
<td></td>
</tr>
<tr>
<td>OCC61</td>
<td>OCC626C</td>
<td>OCC630</td>
<td>77</td>
</tr>
<tr>
<td>OCC62</td>
<td>OCC640</td>
<td>OCC650</td>
<td></td>
</tr>
<tr>
<td>OCC63</td>
<td>OCC660</td>
<td>OCC670</td>
<td>77</td>
</tr>
<tr>
<td>OCC64</td>
<td>OCC680</td>
<td>OCC690</td>
<td></td>
</tr>
<tr>
<td>OCC65</td>
<td>OCC700</td>
<td>OCC71C</td>
<td></td>
</tr>
<tr>
<td>OCC66</td>
<td>OCC720</td>
<td>OCC730</td>
<td>05</td>
</tr>
<tr>
<td>OCC67</td>
<td>OCC740</td>
<td>OCC750</td>
<td></td>
</tr>
<tr>
<td>OCC68</td>
<td>OCC760</td>
<td>OCC770</td>
<td>05</td>
</tr>
<tr>
<td>OCC69</td>
<td>OCC780</td>
<td>OCC790</td>
<td>05</td>
</tr>
<tr>
<td>OCC70</td>
<td>OCC800</td>
<td>OCC810</td>
<td>05</td>
</tr>
<tr>
<td>OCC71</td>
<td>OCC820</td>
<td>OCC830</td>
<td></td>
</tr>
<tr>
<td>OCC72</td>
<td>OCC840</td>
<td>OCC850</td>
<td>05</td>
</tr>
<tr>
<td>OCC73</td>
<td>OCC860</td>
<td>OCC870</td>
<td></td>
</tr>
<tr>
<td>OCC74</td>
<td>OCC880</td>
<td>OCC890</td>
<td>05</td>
</tr>
<tr>
<td>OCC75</td>
<td>OCC900</td>
<td>OCC910</td>
<td>05</td>
</tr>
<tr>
<td>OCC76</td>
<td>OCC920</td>
<td>OCC930</td>
<td></td>
</tr>
<tr>
<td>OCC77</td>
<td>OCC940</td>
<td>OCC950</td>
<td>05</td>
</tr>
<tr>
<td>OCC78</td>
<td>OCC960</td>
<td>OCC970</td>
<td>05</td>
</tr>
<tr>
<td>OCC79</td>
<td>OCC980</td>
<td>OCC990</td>
<td>05</td>
</tr>
<tr>
<td>OCC80</td>
<td>OCC100</td>
<td>OCC101</td>
<td></td>
</tr>
<tr>
<td>OCC81</td>
<td>OCC102</td>
<td>OCC103</td>
<td>05</td>
</tr>
<tr>
<td>OCC82</td>
<td>OCC104</td>
<td>OCC105</td>
<td>05</td>
</tr>
</tbody>
</table>

**IA-02-02-14**
PROCEDURE DIVISION.

OPEN INPUT PERSONNEL-MASTER-FILE

MOVE WS-SWITCH-OFF TO WS-MASTER-FILE-EOF-SWITCH

PERFORM COIC-READ-AND-PRINT THRU 0010-EXIT

IF WS-DEPARTMENT-HOLD EQUAL TO SPACES
    PERFORM 0020-READING-RTN THRU 0020-EXIT
    ELSE
        IF WS-DEPARTMENT-HOLD NOT EQUAL TO 1001-DEPARTMENT
            PERFORM 0030-READING-RTN THRU 0030-EXIT
        ELSE
            WS-DEPARTMENT-HOLD = 0010-DEPARTMENT
            MOVE WS-DEPENDENTS-TAX-RATE TO WS-TAX-RATE
            MOVE WS-0-DEPENDENTS = 0010-DEPARTMENT
            IF IM0 NUMBER-DEPENDENTS = 0010-DEPARTMENT ZERO
                WS-DEPENDENTS-TAX-RATE = 0010-DEPARTMENT
            ELSE
                PERFORM 0050-READING-RTN THRU 0050-EXIT
            END IF
            PERFORM 0060-READING-RTN THRU 0060-EXIT
        END IF
    END IF
END IF

READ PERSONNEL-MASTER-FILE

PERFORM 0010-READING-RTN THRU 0010-EXIT

STOP RUN.
ELSE
IF IMO1-NUMBER-DEPENDENTS LESS THAN 5
    MOVE WS-1-4-DEPENDENTS-TAX-RATE TO WS-TAX-RATE-HOLD
ELSE
    IF IMO1-NUMBER-DEPENDENTS LESS THAN 9
        MOVE WS-5-8-DEPENDENTS-TAX-RATE TO WS-TAX-RATE-HOLD
    ELSE
        IF IMO1-NUMBER-DEPENDENTS LESS THAN 13
            MOVE WS-9-12-DEPENDENTS-TAX-RATE TO WS-TAX-RATE-HOLD
        ELSE
            MOVE WS-L3-44-DEPENDENTS-TAX-RATE TO
            WS-TAX-RATE-HOLD.
ADD 1 IMO1-NUMBER-DEPENDENTS
GIVING WS-FAMILY-SIZE-HOLD.
MULTIPLY IMO1-MONTHLY-PAY BY WS-TAX-RATE-HOLD
GIVING WS-FEDERAL-TAX-HOLD.
MULTIPLY IMO1-MONTHLY-PAY BY WS-FICA-TAX-RATE
GIVING WS-FICA-TAX-HOLD.
SUBTRACT WS-FICA-TAX-HOLD FROM WS-FEDERAL-TAX-HOLD FROM
IMO1-MONTHLY-PAY
GIVING WS-NET-PAY-HOLD
DIVIDE WS-NET-PAY-HOLD BY WS-FAMILY-SIZE-HOLD
GIVING WS-PAY-AVERAGE.
MOVE IMO1-SEC-SEC-NUM TO WS-SEC-SEC-NUM.
MOVE IMO1-NAME TO WS-NAME.
MOVE IMO1-MONTHLY-PAY TO WS-MONTHLY-PAY.
MOVE IMO1-NUMBER-DEPENDENTS TO WS-NUMBER-DEPENDENTS.
MOVE WS-FEDERAL-TAX-HOLD TO WS-FEDERAL-TAX.
MOVE WS-TAX-RATE-HOLD TO WS-TAX-RATE.
MOVE WS-FICA-TAX-HOLD TO WS-FICA-TAX.
MOVE WS-NET-PAY-HOLD TO WS-NET-PAY.
AND WS-NET-PAY-HOLD TO WS-TOTAL-NET-PAY-SUM.
ADD WS-FAMILY-SIZE-HOLD TO WS-TOTAL-FAMILY-SIZE-SUM.
WRITE ORII-PRINTER-REC FROM WS-DETAIL-LINE
AFTER ADVANCING 2 LINES.
CO1C-EXIT. EXIT.
CO20-TOTALS-RTP.
MOVE WS-TOTAL-FAMILY-SIZE-SUM TO WS-TOTAL-FAMILY-SIZE.
MOVE WS-TOTAL-NET-PAY-SUM TO WS-TOTAL-NET-PAY.
WRITE CR01-F10RINT-REC FROM WS-FOOTING-LINE
AFTER ADVANCING 1 LINES.
MOVE ZERCS TO WS-TOTAL-FAMILY-SIZE-SUM
WS-TOTAL-NET-PAY-SUM.
CO193 CC186C 0020-EXIT, EXIT.
CO194 CC186C CO1C-EXIT, EXIT.
CO195 CC1870 CO20-TOTALS-RTP.
MOVE WS-TOTAL-FAMILY-SIZE-SUM TO WS-TOTAL-FAMILY-SIZE.
MOVE WS-TOTAL-NET-PAY-SUM TO WS-TOTAL-NET-PAY.
WRITE CR01-F10RINT-REC FROM WS-FOOTING-LINE
AFTER ADVANCING 1 LINES.
CO220 CC1920 MOVE ZERCS TO WS-TOTAL-FAMILY-SIZE-SUM
CO221 CC193C WS-TOTAL-NET-PAY-SUM.
CO222 CC1940 0020-EXIT, EXIT.
CO223 CC1950 CO30-HEADING-PTA.
MOVE IMO1-CITY-PARTMENT TO WS-DEPARTMENT-HOLD.
WRITE ORII-PRINTER-REC FROM WS-COLUMN-HEADING
AFTER ADVANCING CH1.
CO227 CC1993 0030-EXIT, EXIT.
<table>
<thead>
<tr>
<th>INTRNL NAME</th>
<th>LVL</th>
<th>SOURCE NAME</th>
<th>BASE</th>
<th>DISPL</th>
<th>INTRNL NAME</th>
<th>DEFINITION</th>
<th>USAGE</th>
<th>ROQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNA=1-151</td>
<td>FD</td>
<td>PPRSCMEL-MASTER-FILE</td>
<td>DTF=01</td>
<td>DNA=1-151</td>
<td>DS OCL80</td>
<td>GROUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=1-154</td>
<td>01</td>
<td>IM01-PERSONNEL-MASTER-RCD</td>
<td>BL=1</td>
<td>DNA=1-154</td>
<td>DS 9C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=1-234</td>
<td>02</td>
<td>IM01-SEC-SEC-NUM</td>
<td>BL=1</td>
<td>DNA=1-234</td>
<td>DS 18C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=1-24C</td>
<td>02</td>
<td>IM01-NAME</td>
<td>BL=1</td>
<td>DNA=1-240</td>
<td>DS 2C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=1-279</td>
<td>02</td>
<td>IM01-DEPARTMENT</td>
<td>AL=1</td>
<td>DNA=1-279</td>
<td>DS 5C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=1-304</td>
<td>02</td>
<td>FILLER</td>
<td>AL=1</td>
<td>DNA=1-304</td>
<td>DS 6C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=1-3-20</td>
<td>02</td>
<td>IM01-NUFR-DEPENDENTS</td>
<td>BL=1</td>
<td>DNA=1-320</td>
<td>DS 6C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=1-352</td>
<td>02</td>
<td>IPC1-MONTHLY-PAY</td>
<td>AL=1</td>
<td>DNA=1-352</td>
<td>DS 6C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=1-386</td>
<td>02</td>
<td>FILLER</td>
<td>AL=1</td>
<td>DNA=1-386</td>
<td>DS 8C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=1-397</td>
<td>FD</td>
<td>PRINTER</td>
<td>DTF=02</td>
<td>DNA=1-397</td>
<td>DS 133C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=1-428</td>
<td>01</td>
<td>ORD1-PRINTER-RCD</td>
<td>BL=2</td>
<td>DNA=1-428</td>
<td>DS 14C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=1-444</td>
<td>77</td>
<td>WS-MASTER-FILE-EOF-SWITCH</td>
<td>BL=3</td>
<td>DNA=1-444</td>
<td>DS 15C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=2-207</td>
<td>77</td>
<td>WS-TOTAL-FAMILY-SIZE-SUM</td>
<td>BL=3</td>
<td>DNA=2-207</td>
<td>DS 16C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=2-237</td>
<td>77</td>
<td>WS-FOH-FOH</td>
<td>BL=3</td>
<td>DNA=2-237</td>
<td>DS 17C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=2-307</td>
<td>77</td>
<td>WS-FOH-FOH</td>
<td>BL=3</td>
<td>DNA=2-307</td>
<td>DS 18C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=2-337</td>
<td>77</td>
<td>WS-FOH-FOH</td>
<td>BL=3</td>
<td>DNA=2-337</td>
<td>DS 19C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=2-402</td>
<td>77</td>
<td>WS-FOH-FOH</td>
<td>BL=3</td>
<td>DNA=2-402</td>
<td>DS 20C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=2-440</td>
<td>77</td>
<td>WS-FOH-FOH</td>
<td>BL=3</td>
<td>DNA=2-440</td>
<td>DS 21C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-487</td>
<td>77</td>
<td>WS-FOH-FOH</td>
<td>BL=3</td>
<td>DNA=3-487</td>
<td>DS 22C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-300</td>
<td>01</td>
<td>WS-COLUMN-HEADING</td>
<td>BL=3</td>
<td>DNA=3-300</td>
<td>DS 23C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-330</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-330</td>
<td>DS 24C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-444</td>
<td>32</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-444</td>
<td>DS 25C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-464</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-464</td>
<td>DS 26C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-674</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-674</td>
<td>DS 27C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-125</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-125</td>
<td>DS 28C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-114</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-114</td>
<td>DS 29C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-163</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-163</td>
<td>DS 30C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-182</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-182</td>
<td>DS 31C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-201</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-201</td>
<td>DS 32C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-270</td>
<td>01</td>
<td>WS-DETAIL-LINE</td>
<td>BL=3</td>
<td>DNA=3-270</td>
<td>DS 33C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-250</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-250</td>
<td>DS 34C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-260</td>
<td>02</td>
<td>WS-SEC-SEC-UM</td>
<td>BL=3</td>
<td>DNA=3-260</td>
<td>DS 35C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-292</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-292</td>
<td>DS 36C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-312</td>
<td>02</td>
<td>WS-NAME</td>
<td>BL=3</td>
<td>DNA=3-312</td>
<td>DS 37C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-325</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-325</td>
<td>DS 38C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-348</td>
<td>02</td>
<td>WS-MONTHLY-PAY</td>
<td>BL=3</td>
<td>DNA=3-348</td>
<td>DS 39C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-343</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>DNA=3-343</td>
<td>DS 40C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-438</td>
<td>02</td>
<td>WS-NUFR-DEPENDENTS</td>
<td>BL=3</td>
<td>DNA=3-438</td>
<td>DS 41C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA=3-458</td>
<td>02</td>
<td>WS-PAY-AVERAGE</td>
<td>BL=3</td>
<td>DNA=3-458</td>
<td>DS 42C</td>
<td>DISP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MEMORY MAP

TGT  0249

SAVE AREA  0C540
SWITCH  0C550
TALLY  0C554
SORT SAVE  0C559
ENTRY-SAVE  0C55C
SORT CORE SIZE  005A0
ASTIC-REELS  005A4
SORT RET  0C5A6
WORKING CELLS  005A9
SORT FILE SIZE  0C668
SORT MORE SIZE  0C66C
PGT-VN TBL  0C6F0
TGT-VN TBL  0C6F4
SORTAB ADDRESS  0C6FE
LENGTH OF VN TBL  0C6FC
LENGTH OF SORTAP  0C6FE
PGP ID  006F0
INIT  0C6F8
UPSI SWITCHES  0C6FC
OVERFLOW CELLS  0C704
BL CELLS  00704
DTFACR CELLS  0C710
TEMP STORAGE  0C71A
TEMP STOPAGE-2  0C728
TEMP STORAGE-3  0C738
TEMP STORAGE-4  0078A
ALL CELLS  0C738
VLC CELLS  0C73C
SRL CELLS  0C73C
INDEX CELLS  0C73C
SADAP CELLS  0073C
CNTRL CELLS  0C73C
PFNCTL CELLS  0073C
PFN SAV CELLS  0C73C
VN CELLS  0C750
SAVE AREA =2  0075C
XSASW CELLS  0C75C
XSA CELLS  0C75C
PARAM CELLS  0075C
RPTS SAV AREA  0C75C
ICPTC CTOR  0C75C
ICPTR CELLS  0C75C

LITERAL FOOL (HEX)

00700 LIT+01  CFC0C81C 3C1C4020 58202021 2C642020 40202120 F3202120
C078A LIT+24)  0000C0C0 0C0C0004 585AC206 07C5D40 585AC206 03D6E2C5

IA-02-02-14

312
<table>
<thead>
<tr>
<th>REGISTER ASSIGNMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>REG A BL = 3</td>
</tr>
<tr>
<td>REG 7 BL = 1</td>
</tr>
<tr>
<td>REG A BL = 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>137</td>
<td>START</td>
<td>EQU</td>
</tr>
<tr>
<td>CCCCC0</td>
<td>L</td>
<td>1,1CA(0,13)</td>
</tr>
<tr>
<td>CCCCCC</td>
<td>SH</td>
<td>1,08D(0,12)</td>
</tr>
<tr>
<td>CCCCC8</td>
<td>NI</td>
<td>010011,X<em>RF</em></td>
</tr>
<tr>
<td>CCCCC9</td>
<td>LA</td>
<td>1,08D(0,12)</td>
</tr>
<tr>
<td>CCCCC7</td>
<td>L</td>
<td>0,1CD(0,13)</td>
</tr>
<tr>
<td>CCCCC6</td>
<td>LR</td>
<td>4,0</td>
</tr>
<tr>
<td>CCCCC5</td>
<td>BAL</td>
<td>15,0</td>
</tr>
<tr>
<td>CCCCC4</td>
<td>ST</td>
<td>0,09D(0,15)</td>
</tr>
<tr>
<td>CCCCC3</td>
<td>BAL</td>
<td>0,00D(0,15)</td>
</tr>
<tr>
<td>CCCCC2</td>
<td>DC</td>
<td>X'00000000'</td>
</tr>
<tr>
<td>CCCCC1</td>
<td>SVC</td>
<td>2</td>
</tr>
<tr>
<td>CCCCC0</td>
<td>LA</td>
<td>1,08D(0,12)</td>
</tr>
<tr>
<td>CCCCC9</td>
<td>L</td>
<td>0,1CD(0,13)</td>
</tr>
<tr>
<td>CCCCC8</td>
<td>LR</td>
<td>4,0</td>
</tr>
<tr>
<td>CCCCC7</td>
<td>BCR</td>
<td>0,0</td>
</tr>
<tr>
<td>CCCCC6</td>
<td>BALR</td>
<td>15,0</td>
</tr>
<tr>
<td>CCCCC5</td>
<td>ST</td>
<td>0,09D(0,15)</td>
</tr>
<tr>
<td>CCCCC4</td>
<td>RAL</td>
<td>0,00D(0,15)</td>
</tr>
<tr>
<td>CCCCC3</td>
<td>DC</td>
<td>X'00000000'</td>
</tr>
<tr>
<td>CCCCC2</td>
<td>SVC</td>
<td>2</td>
</tr>
<tr>
<td>CCCCC1</td>
<td>EQU</td>
<td></td>
</tr>
<tr>
<td>CCCCC0</td>
<td>L</td>
<td>1,1CA(0,13)</td>
</tr>
<tr>
<td>CCCCC9</td>
<td>L</td>
<td>0,20D(0,13)</td>
</tr>
<tr>
<td>CCCCC8</td>
<td>ST</td>
<td>0,1F4(0,13)</td>
</tr>
<tr>
<td>CCCCC7</td>
<td>L</td>
<td>0,01C(0,12)</td>
</tr>
<tr>
<td>CCCCC6</td>
<td>ST</td>
<td>0,20B(0,13)</td>
</tr>
<tr>
<td>CCCCC5</td>
<td>EQU</td>
<td></td>
</tr>
<tr>
<td>CCCCC4</td>
<td>L</td>
<td>2,02D(0,12)</td>
</tr>
<tr>
<td>CCCCC3</td>
<td>CLC</td>
<td>00011,13H,025(6)</td>
</tr>
<tr>
<td>CCCCC2</td>
<td>PCR</td>
<td>8,2</td>
</tr>
<tr>
<td>CCCCC1</td>
<td>L</td>
<td>1,004(0,12)</td>
</tr>
</tbody>
</table>

IA-02-02-14
### DATA NAMES

<table>
<thead>
<tr>
<th>NAME</th>
<th>DEFN</th>
<th>REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFSCHWFL-MASTER-FILE</td>
<td>0012</td>
<td>0137 00142 00146 00146</td>
</tr>
<tr>
<td>IMCI-SEC-SEC-NUM</td>
<td>0021</td>
<td>C182</td>
</tr>
<tr>
<td>IMCI-NAME</td>
<td>C822</td>
<td>C183</td>
</tr>
<tr>
<td>IMCI-DEPARTMENT</td>
<td>C023</td>
<td>C114 00204</td>
</tr>
<tr>
<td>IMCI-MEMBER-DEPENDENTS</td>
<td>C005</td>
<td>0157 0160 0163 0166 0171 0185</td>
</tr>
<tr>
<td>IMCI-MONTHLY-PAY</td>
<td>C064</td>
<td>C173 0175 0177 0184</td>
</tr>
<tr>
<td>PRINTER</td>
<td>C014</td>
<td>C137 0137 0142 0192 0198 0205</td>
</tr>
<tr>
<td>IMCI-PRINTER-RCN</td>
<td>C03C</td>
<td>C192 0192 0192 0194 0198 0198 0198 0198 0205 0205 0205 0205 0205</td>
</tr>
<tr>
<td>WS-MASTER-FILE-ECF-SWITCH</td>
<td>C032</td>
<td>C110 0140 0149</td>
</tr>
<tr>
<td>WS-TOTAL-FAMILY-SIZE-SUM</td>
<td>C033</td>
<td>C110 0116 0200</td>
</tr>
<tr>
<td>WS-TOTAL-NET-PAY-SUM</td>
<td>C036</td>
<td>C190 0197 0200</td>
</tr>
<tr>
<td>WS-DEPARTMENT-HCLO</td>
<td>CC39</td>
<td>C151 0151 0151 0154 0204</td>
</tr>
<tr>
<td>WS-FEDERAL-TAX-HCLO</td>
<td>C041</td>
<td>C173 0173 0177 0186</td>
</tr>
<tr>
<td>WS-FICA-TAX-HCLO</td>
<td>C043</td>
<td>C175 0177 0186</td>
</tr>
<tr>
<td>WS-NET-PAY-HCLO</td>
<td>C045</td>
<td>C177 0177 0180 0199 03190</td>
</tr>
<tr>
<td>WS-FAMILY-SIZE-HCLO</td>
<td>C047</td>
<td>C171 0180 0191</td>
</tr>
<tr>
<td>WS-TAX-FAT-HCLO</td>
<td>C049</td>
<td>0158 0161 0164 0167 0169 0169 0169 0169</td>
</tr>
<tr>
<td>WS-FICA-TAX-RATE</td>
<td>C051</td>
<td>01175</td>
</tr>
<tr>
<td>WS-O-DEPENDENTS-TAX-RATE</td>
<td>C054</td>
<td>C159</td>
</tr>
<tr>
<td>WS-1-4-DEPENDENTS-TAX-RATE</td>
<td>CC37</td>
<td>C111</td>
</tr>
<tr>
<td>WS-5-9-DEPENDENTS-TAX-RATE</td>
<td>CC60</td>
<td>C164</td>
</tr>
<tr>
<td>WS-10-12-DEPENDENTS-TAX-RATE</td>
<td>C043</td>
<td>C167</td>
</tr>
<tr>
<td>WS-13-99-DEPENDENTS-TAX-RATE</td>
<td>CC66</td>
<td>C169</td>
</tr>
<tr>
<td>WS-SWITCH-OFF</td>
<td>CC71</td>
<td>C139</td>
</tr>
<tr>
<td>WS-COLUMN-FADING</td>
<td>C073</td>
<td>CG705 00025</td>
</tr>
<tr>
<td>WS-DETAIL-LINE</td>
<td>CC94</td>
<td>C192 0192</td>
</tr>
<tr>
<td>WS-SOC-SEC-NUM</td>
<td>CC57</td>
<td>C182</td>
</tr>
<tr>
<td>WS-KAFF</td>
<td>C0100</td>
<td>C183</td>
</tr>
<tr>
<td>WS-MONTHLY-PAY</td>
<td>C123</td>
<td>C184</td>
</tr>
<tr>
<td>WS-NUM1-FFCT-DEPENDENTS</td>
<td>C0106</td>
<td>C185</td>
</tr>
<tr>
<td>WS-PAY-AVERAGE</td>
<td>C0109</td>
<td>C180</td>
</tr>
<tr>
<td>WS-FEDERAL-TAX</td>
<td>CC112</td>
<td>C186</td>
</tr>
<tr>
<td>WS-TAX-RATE</td>
<td>0115</td>
<td>C187</td>
</tr>
<tr>
<td>WS-FICA-TAX</td>
<td>C0118</td>
<td>C188</td>
</tr>
<tr>
<td>WS-NET-PAY</td>
<td>00121</td>
<td>C189</td>
</tr>
<tr>
<td>WS-FOOTING-LINE</td>
<td>C0124</td>
<td>C199</td>
</tr>
<tr>
<td>WS-TOTAL-FAMILY-SIZE</td>
<td>C0130</td>
<td>C196</td>
</tr>
<tr>
<td>WS-TOTAL-NET-PAY</td>
<td>C0133</td>
<td>C197</td>
</tr>
</tbody>
</table>

### PROCEDURE NAMES

<table>
<thead>
<tr>
<th>NAME</th>
<th>DEFN</th>
<th>REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010-READ-ANC-PRINT</td>
<td>00145</td>
<td>C140</td>
</tr>
<tr>
<td>0010-EXIT</td>
<td>00144</td>
<td>C140 00150</td>
</tr>
<tr>
<td>0020-TOTALS-RTN</td>
<td>C195</td>
<td>C149 00155</td>
</tr>
<tr>
<td>0020-EXIT</td>
<td>00202</td>
<td>C148 00155</td>
</tr>
</tbody>
</table>

IA-02-02-14
UNREFERENCED SYMBOLS

002 WARNING ADDRESS CONSTANT
<table>
<thead>
<tr>
<th>SSN</th>
<th>NAME</th>
<th>MONTHLY PAY</th>
<th>NO OF DEPS</th>
<th>PAY AVE</th>
<th>FED TAX</th>
<th>TAX RATE</th>
<th>FICA TAX</th>
<th>NET PAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>724764925</td>
<td>GRUNDA GRYEY JR</td>
<td>$564.00</td>
<td>12</td>
<td>$45.44</td>
<td>08 %</td>
<td>$32.94</td>
<td>$489.62</td>
<td></td>
</tr>
<tr>
<td>639167030</td>
<td>HADDI REY</td>
<td>$354.00</td>
<td>9</td>
<td>$31.84</td>
<td>08 %</td>
<td>$23.08</td>
<td>$343.08</td>
<td></td>
</tr>
<tr>
<td>257603340</td>
<td>CHARLEY HELEN WILLI</td>
<td>$344.00</td>
<td>12</td>
<td>$31.41</td>
<td>12 %</td>
<td>$19.95</td>
<td>$282.77</td>
<td></td>
</tr>
<tr>
<td>35411904</td>
<td>RITCHISON RENNIE L</td>
<td>$569.00</td>
<td>0</td>
<td>$421.46</td>
<td>20 %</td>
<td>$32.94</td>
<td>$421.46</td>
<td></td>
</tr>
</tbody>
</table>

Department Total Family Size 1 and Total Net Pay $1115.47

Department Total Family Size 3 and Total Net Pay $421.46
SSN | NAME | MONTHLY PAY | NO OF DEPS | PAY AVE | FED TAX | TAX RATE | FICA TAX | NET PAY

05031 PROGRAM CHECK INTERUPTION - HEK LOCATION 0C59P2 - CONDITION CODE 3 - DATA EXCEPTION
05001 JCR DUMPPE3 CANCELED

IA-02-02-14
INSTRUCTIONS TO THE STUDENT:

This booklet contains three (3) COBOL Source program listings, each with an Associated Compiler Error Diagnostic Listing.

You are to correct the syntax errors by;

1. Circle the portion of the source statement in error,
2. pencil in the correction to be made on the source listing.

When you correct all the errors in the first practical exercise turn to the back of this booklet and check your answers against the school solution. If you have any questions concerning this exercise see your instructor for assistance.

Follow this procedure for all three practical exercises in this booklet.
IDENTIFICATION DIVISION.

PROGRAM-ID. COBSYN1.

AUTHOR. SP6 DM GROSS, SOFTWARE.

INSTALLATION. GSD, USAIA, FT HARRISON, IN 46216.

DATE-WRITTEN. MAR 3 1978.

DATE-Compiled. JUN 15, 1978

SECURITY. UNCLASSIFIED.

*REMARKS. THIS PROGRAM LISTS ALL ORDERS RECEIVED EACH DAY. IT LISTS THE PART NUMBER, CUSTOMER NAME, AND CUSTOMER ADDRESS.

WS-ORDER-EOF-SW IS USED IN THE PROCEDURE DIVISION TO SIGNIFY WHEN THE END OF FILE CONDITION HAS BEEN REACHED.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-360-H40.

OBJECT-COMPUTER. IBM-360-H40.

SPECIAL-NAMES.

GO1 IS TOP-OF-PAGE.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT IM-ORDER-FILE
ASSIGN TO UT-S-SYS006.

SELECT OR-ORDERS-LISTING
ASSIGN TO UT-S-SYS005.

DATA DIVISION.

FILE SECTION.

FD IM-ORDER-FILE,
BLOCK CONTAINS 1 RECORDS,
RECORD CONTAINS 80 CHARACTERS,
LABEL RECORD IS STANDARD,
DATA RECORD IS IM01-ORDER-RC.

01 IM01-ORDER-RC.
05 IM01-PART-NO PIC 9(4).
05 IM01-CUSTOMER-NAME PIC X(18).
05 IM01-CUSTOMER-ADDRESS PIC X(40).
05 IM01-UNIT-OF-ISSUE PIC XXX.
05 IM01-UNIT-OF-ORDER PIC 999.
05 IM01-PRICE-ITEM PIC 9(13) V99.
05 IM01-CUSTOMER-CODE PIC X.
05 FILLER PIC X(31).

FD OR-ORDERS-LISTING,
BLOCK CONTAINS 1 RECORDS,
RECORD CONTAINS 133 CHARACTERS,
LABEL RECORD IS STANDARD,
DATA RECORD IS OR01-ORDER-LINE.

01 OR01-ORDER-LINE
WORKING-STOREAGE SECTION.

77 WS-ORDER-EOF-SW VALUE 'OFF'.

01 WS-HEADING-LINE.
05 FILLER PIC X(33).

VALUE SPACES.

IA-01-03-03
00055 05 FILLER VALUE 'PART NO'. PIC X(7)
00056 05 FILLER VALUE SPACES. PIC X(11)
00057 05 FILLER VALUE 'CUSTOMER NAME'. PIC X(13)
00058 05 FILLER VALUE SPACES. PIC X(12)
00059 05 FILLER VALUE 'CUSTOMER ADDRESS'. PIC X(16)
00060 05 FILLER VALUE SPACES. PIC X(41)
00061 05 FILLER PIC X(35)
00062 01 WS-DETAIL-LINE.
00063 05 FILLER PIC X(35)
00064 05 WS-DL-PART-NO PIC 9(4).
00065 05 FILLER PIC 1(X0)
00066 05 WS-DL-CUSTOMER-NAME PIC X(18).
00067 05 FILLER PIC X(8)
00068 05 WS-DL-CUSTOMER-ADDRESS PIC X(40).
00069 05 FILLER PIC X(40)
00070 05 WS-DL-CUSTOMER-NAME PIC X(18).
00071 05 FILLER PIC X(8)
00072 05 WS-DL-CUSTOMER-ADDRESS PIC X(18).
00073 05 FILLER PIC X(40)
00074 05 WS-DL-CUSTOMER-NAME PIC X(18).
00075 05 FILLER PIC X(8)
00076 05 WS-DL-CUSTOMER-ADDRESS PIC X(18).
00077 05 FILLER PIC X(40)
00078 05 WS-DL-CUSTOMER-NAME PIC X(18).
00079 05 FILLER PIC X(8)
00080 05 WS-DL-CUSTOMER-ADDRESS PIC X(18).
00081 05 FILLER PIC X(40)
00082 PROCEDURE DIVISION.
00083 0010 DRIVER.
00084 OPEN INPUT IM-ORDER-FILE.
00085 0010-EXIT.
00086 0010-EXIT.
00087 0020 READ-LIST-RTN.
00088 0020 READ-LIST-RTN.
00089 0020 READ-LIST-RTN.
00090 0020 READ-LIST-RTN.
00091 0020 READ-LIST-RTN.
00092 0020 READ-LIST-RTN.
00093 0020 READ-LIST-RTN.
00094 0020 READ-LIST-RTN.
00095 0020 READ-LIST-RTN.
00096 0020 READ-LIST-RTN.
00097 0020 READ-LIST-RTN.
00098 0020 READ-LIST-RTN.
00099 0020 READ-LIST-RTN.
00100 0020 READ-LIST-RTN.
00101 0020-EXIT.
00102 0020-EXIT.

PROCEDURE DIVISION.
00080 0010 DRIVER.
00081 0010-EXIT.
00082 0010-EXIT.
00083 0010-EXIT.
00084 0010-EXIT.
00085 0010-EXIT.
00086 0010-EXIT.
00087 0010-EXIT.
00088 0010-EXIT.
00089 0010-EXIT.
00090 0010-EXIT.
00091 0010-EXIT.
00092 0010-EXIT.
00093 0010-EXIT.
00094 0010-EXIT.
00095 0010-EXIT.
00096 0010-EXIT.
00097 0010-EXIT.
00098 0010-EXIT.
00099 0010-EXIT.
00100 0010-EXIT.
00101 0010-EXIT.
00102 0010-EXIT.

PROCEDURE DIVISION.
00080 0010 DRIVER.
00081 OPEN INPUT IM-ORDER-FILE.
00082 OUTPUT OR-ORDERS-LISTING.
00083 WRITE OR01-ORDER-LINE FROM WS-HEADING-LINE.
00084 AFTER ADVANCING TOP-OF-PAGE.
00085 PERFORM 0020-READ-LIST-RTN THRU 0020-EIXT UNTIL WS-ORDER-EOF-SW EQUAL 'ON'.
00086 CLOSE IM-ORDER-FILE.
00087 OR-ORDERS-LISTING.
00088 STOP RUN.
00089 STOP RUN.
00090 STOP RUN.
00091 STOP RUN.
00092 STOP RUN.
00093 STOP RUN.
00094 STOP RUN.
00095 STOP RUN.
00096 STOP RUN.
00097 STOP RUN.
00098 STOP RUN.
00099 STOP RUN.
00100 STOP RUN.
00101 STOP RUN.
00102 STOP RUN.

PROCEDURE DIVISION.
00080 0010 DRIVER.
00081 OPEN INPUT IM-ORDER-FILE.
00082 OUTPUT OR-ORDERS-LISTING.
00083 WRITE OR01-ORDER-LINE FROM WS-HEADING-LINE.
00084 AFTER ADVANCING TOP-OF-PAGE.
00085 PERFORM 0020-READ-LIST-RTN THRU 0020-EIXT UNTIL WS-ORDER-EOF-SW EQUAL 'ON'.
00086 CLOSE IM-ORDER-FILE.
00087 OR-ORDERS-LISTING.
00088 STOP RUN.
00089 STOP RUN.
00090 STOP RUN.
00091 STOP RUN.
00092 STOP RUN.
00093 STOP RUN.
00094 STOP RUN.
00095 STOP RUN.
00096 STOP RUN.
00097 STOP RUN.
00098 STOP RUN.
00099 STOP RUN.
00100 STOP RUN.
00101 STOP RUN.
<table>
<thead>
<tr>
<th>CARD</th>
<th>ERROR MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>IKF10041-E</td>
</tr>
<tr>
<td>23</td>
<td>IKF21461-W</td>
</tr>
<tr>
<td>71</td>
<td>IKF20391-C</td>
</tr>
<tr>
<td>71</td>
<td>IKF21291-C</td>
</tr>
<tr>
<td>79</td>
<td>IKF10431-W</td>
</tr>
<tr>
<td>85</td>
<td>IKF30011-E</td>
</tr>
<tr>
<td>96</td>
<td>IKF30011-E</td>
</tr>
<tr>
<td>99</td>
<td>IKF40031-E</td>
</tr>
</tbody>
</table>
**ENVIRONMENT DIVISION.**
**CONFIGURATION SECTION.**

**SOURCE-COMPUTER.** IBM-360-H40.

**OBJECT-COMPUTER.** IBM-360-H40.

**SPECIAL-NAMES.**

**C01** IS TOP-OF-PAGE.

**INPUT-OUTPUT SECTION.**

**FILE-CONTROL.**

- SELECT IM-PERSONNEL-MASTER-FILE
  ASSIGN TO TU-S-SYS006.

- SELECT OR-PRINTER
  ASSIGN TO UT-S-SYS005.

**DATA DIVISION.**

**FILE SECTION.**

- FD IM-PERSONNEL-MASTER-FILE
  BLOCK CONTAINS 1 RECORDS,
  RECORD CONTAINS 80 CHARACTERS,
  LABEL RECORD IS STANDARD,
  DATA RECORD IS IM01-PERSONNEL-MASTER-RCD.

- 01 IM01-PERSONNEL-MASTER-RCD.

  05 IM01-BATTLAION  PIC XX.
  05 IM01-COMPANY  PIC X.
  05 IM01-SSAN  PIC 9(9).
  05 IM01-NAME  PIC X(18).
  05 IM01-PAY-GRADE  PIC XY.
  05 IM01-RANK  PIC X(3).
  05 IM01-RANK  PIC X(6).
  05 IM01-DATE-OF-ENTRY  PIC X(6).
  05 IM01-MONTHS-OF-SERVICE  PIC 9(3).
  05 IM01-NUMBER-OF-DEPENDENTS  PIC 99.
  05 IM01-PREVIOUS-ASSIGN-INFO  PIC X(28).

- FD OR-PRINTER.

  01 OR01-PRINTER-RCD  PIC X(13X).

  WORKING-STORAGE SECTION.

  77 WS-MASTER-FILE-EOF-SWITCH  PIC XXX.

  01 WS-HEADER-LINE.

  05 FILLER  VALUE 'OFF' PIC X(53).

  05 FILLER  VALUE SPACES. PIC X(5).
PROCEDURE DIVISION.

0010-DRIVER.
OPEN INPUT IM-PERSONNEL-MASTER-FILE
OUTPUT OR-PRINTER.
WRITE OR01-PRINTER-RCD FROM WS-HEADING-LINE
AFTER ADVANCING TOP-OF-PAGE.
PERFORM 0020-READ-LIST-RTN THRU 0020-EXIT
UNTIL WS-MASTER-FILE-EOF-SWITCH EQUAL 'ON'.
CLOSE IM-PERSONNEL-MASTER-FILE.
OR-PRINTER.

0010-EXIT.
STOP RUN.

0020-READ-LIST-RTN.
READ IM-PERSONNEL-MASTER-FILE
AT END
MOVE 'ON' TO WS-MASTER-FILE-EOF-SWITCH
GO TO 0020-EXIT.

IF IM01-MONTHS-OF-SERVICE GREATER THAN 75 THEN
MOVE IM01-NAME TO WS-DL-NAME
MOVE IM01-RANK TO WS-DL-RANK
MOVE IM01-SSAN TO WS-DL-SSAN
MOVE WS-DL-MONTHS-OF-SERVICE TO IM01-MONTHS-OF-SERVICE
WRITE OR01-PRINTER-RCD FROM WS-DETAIL-LINE
AFTER ADVANCING 2 LINES.

0020-EXIT.
EXIT.
CARD   ERROR MESSAGE
1      IKF1291-C   ID DIV. HEADER EXTRANEOUS, MISSING OR MISPLACED. ONE ASSUMED PRESENT.
5      IKF10871-W  DATE ' SHOULD NOT BEGIN A-MARGIN.
20     IKF1551-W   DEVICE CLASS INVALID IN SYSTEM-NAME. SKIPPING TO NEXT FIELD.
35     IKF20391-C  PICTURE CONFIGURATION ILLEGAL. PICTURE CHANGED TO 9 UNLESS USAGE IS 'DISPLAY-ST',
           THEN L(16)BDZ9BDZ9.
19     IKF21461-W  RECORD SIZE IN RECORD-CONTAINS CLAUSE DISAGREES WITH COMPUTED RECORD SIZE. USING
           MAXIMUM COMPUTED SIZE.
47     IKF20391-C  PICTURE CONFIGURATION ILLEGAL. PICTURE CHANGED TO 9 UNLESS USAGE IS 'DISPLAY-ST',
           THEN L(16)BDZ9BDZ9.
51     IKF10431-W  END OF SENTENCE SHOULD PRECEDE 01, ASSUMED PRESENT.
21     IKF21461-W  RECORD SIZE IN RECORD-CONTAINS CLAUSE DISAGREES WITH COMPUTED RECORD SIZE. USING
           MAXIMUM COMPUTED SIZE.
54     IKF21261-C  VALUE CLAUSE LITERAL TOO LONG, TRUNCATED TO PICTURE SIZE.
58     IKF10431-W  END OF SENTENCE SHOULD PRECEDE 01, ASSUMED PRESENT.
72     IKF10041-E   INVALID WORD VALUE. SKIPPING TO NEXT RECOGNIZABLE WORD.
79     IKF10071-W   SLASH NOT PRECEDED BY A SPACE. ASSUME SPACE.
79     IKF30011-E   002 NOT DEFINED. STATEMENT DISCARDED.
79     IKF10071-W   MINUS SIGN NOT PRECEDED BY A SPACE. ASSUME SPACE.
79     IKF30011-E   READ-LIST-RTN NOT DEFINED.
82     IKF10861-W   OR-PRINTER SHOULD BEGIN A-MARGIN.
95     IKF40521-E   DMM=1-358 (IEDI) MAY NOT BE TARGET FIELD FOR DMM=2-283 (NE) IN MOVE STATEMENT, AND IS DISCARDED.
IDENTIFICATION DIVISION.
PROGRAM-ID. COBSYN3.
AUTHOR. SP6 DW GROSS.
INSTALLATION. CSD, USAIA, FT HARRISON, IN 46216.
SECURITY. UNCLASSIFIED.
*REMARKS. THIS PROGRAM PRODUCES THE DEPENDENT INFORMATION REPORT. IT LISTS ALL PERSONNEL IN THE BATTALION WITH 5 OR MORE DEPENDENTS BY NAME, RANK, COMPANY, AND SIZE OF FAMILY.
WS-MASTER-FILE-EOF-SWITCH IS USED TO SIGNIFY WHEN THE END OF FILE CONDITION HAS BEEN ENCOUNTERED.
WS-TOTAL-DEPENDENTS IS USED TO ACCUMULATE THE DEPENDENTS FOR THE PURPOSE OF PRINTING IT IN THE TOTAL LINE.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE COMPUTER. IBM-360-H40.
OBJECT-COMPUTER. IBM-360-H40.
SPECIAL-NAMES.
COI IS TOP-OF-PAGE.
INPUT-OUTPUT SECTION.
FILE CONTROL.
SELECT IM-PERSONNEL-MASTER-FILE
ASSIGN TO UT-S-SYS006.
SELECT OR-PRINTER
ASSIGN TO UT-S-SYS005.
DATA DIVISION.
FILE SECTION.
FD IM-PERSONNEL-MASTER-FILE,
BLOCK CONTAINS 1 RECORDS,
RECORD CONTAINS 80 CHARACTERS,
LABEL RECORD IS STANDARD,
DATA RECORD IS IM01-PERSONNEL-MASTER-RCD.
FD IM01-PERSONNEL-MASTER-RCD
05 IM01-BATTALION PIC XX.
05 IM01-COMPANY PIC X.
05 IM01-SSAN PIC 9(9).
05 IM01-NAME PIC X(18).
05 IM01-PAY-GRADE PIC XX.
05 IM01-RANK PIC XXX.
05 IM01-ETS PIC X(6).
05 IM01-DATE-OF-ENTRY PIC X(6).
05 IM01-MONTHS-OF-SERVICE PIC 999.
05 IM01-NUMBER-OF-DEPENDENTS PIC XX.
05 IM01-PREVIOUS-ASSIGN-INFO PIC X(28).
FD OR-PRINTER.
BLOCK CONTAINS 1 RECORDS,
RECORD CONTAINS 133 CHARACTERS,
LABEL RECORD IS STANDARD,
DATA RECORD IS ORO1-PRINTER-RCD.

01 ORO1-PRINTER-RCD
WORKING-STORAGE SECTION.

77 WS-LINE-COUNTER
VALUE ZEROS.

77 WS-TOTAL-DEPENDENTS
VALUE ZEROS.

77 WS-MASTER-FILE-EOF-SWITCH
VALUE 'OFF'.

01 WS-HEADING-LINE-1.
  05 FILLER VALUE SPACES.
  05 FILLER VALUE 'DEPENDENT INFORMATION REPORT'.
  05 FILLER VALUE SPACES.

01 WS-HEADING-LINE-2.
  05 FILLER VALUE SPACES.
  05 FILLER VALUE 'NAME'.
  05 FILLER VALUE SPACES.
  05 FILLER VALUE 'RANK'.
  05 FILLER VALUE SPACES.

01 WS-DETAIL-LINE.
  05 FILLER VALUE SPACES.
  05 WS-DL-NAMENC
VALUE SPACES.
  05 WS-DL-RANK
VALUE SPACES.
  05 WS-DL-COMPANY
VALUE SPACES.
  05 WS-DL-FAMILY-SIZE
VALUE SPACES.

01 WS-TOTAL-LINE.
  05 FILLER VALUE SPACES.
  05 FILLER VALUE 'TOTAL NUMBER OF DEPENDENTS WITHIN'.
  05 FILLER VALUE 'THE BATTALION IS'.

WORKING STORAGE SECTION.

77 WS LINE COUNTER
PIC Z9.

77 WS TOTAL DEPENDENTS
PIC ZZ9.

77 WS MASTER FILE EOF SWITCH
PIC XXX.

331
PROCEDURE DIVISION.

0010-DRIVER.
OPEN INPUT IM-PERSONNEL-MASTER-FILE
OUTPUT OR-PRINTER.
PERFORM 0030-HEADING-RTN THRU 0030-EXIT.
PERFORM 0020-READ-WRITE-RTN THRU 0020-EXIT
UNTIL WS-MASTER-FILE-EOF-SWITCH EQUAL 'ON'.
MOVE WS-TOTAL-DEPENDENTS TO WS-TL-NUMBER-OF-DEPENDENTS.
WRITE OR01-PRINTER-RCD FROM WS-TOTAL-LINE
AFTER ADVANCING 3 LINES.
CLOSE IM-PERSONNEL-MASTER-FILE,
OR-PRINTER.

0010-EXIT.
EXIT.

0020-READ-WRITE-RTN.
READ IM-PERSONNEL-MASTER-FILE
AT END
MOVE 'ON' TO WS-MASTER-FILE-EOF-SWITCH
GO TO 0020-EXIT.
ADD IM01-NUMBER-OF-DEPENDENTS TO WS-TOTAL-DEPENDENTS.
IF IM01-NUMBER-OF-DEPENDENTS LESS THAN 4
THEN
GO TO 0020-EXIT.
IF WS-LINE-COUNTER GREATER THAN 15
THEN
PERFORM 0030-HEADING-RTN THRU 0030-EXIT.
MOVE IM01-NAME TO WS-DL-NAME.
MOVE IM01-RANK TO WS-DL-RANK.
MOVE IM01-COMPANY TO WS-DL-COMPANY.
ADD 1, IM01-NUMBER-OF-DEPENDENTS GIVING WS-DL-FAMILY-SIZE.
WRITE OR01-PRINTER-RCD FROM WS-DETAIL-LINE
AFTER ADVANCING 2 LINES.
ADD 1 TO WS-LINE-COUNTER.

0010-EXIT.
EXIT.

0030-HEADING-RTN.
MOVE 0 TO WS-LINE-COUNTER.
WRITE OR01-PRINTER-RCD FROM WS-HEADING-LINE-1
AFTER ADVANCING TOP-OF-PAGE.
WRITE OR01-PRINTER-RCD FROM WS-HEADING-LINE-2
AFTER ADVANCING 3 LINES.

0030-EXIT.
EXIT.
<table>
<thead>
<tr>
<th>CARD</th>
<th>ERROR MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>IKF1087I-W</td>
</tr>
<tr>
<td></td>
<td>PROGRAM 4 SHOULD NOT BEGIN A-MARGIN.</td>
</tr>
<tr>
<td>2</td>
<td>IKF1087I-E</td>
</tr>
<tr>
<td></td>
<td>PROGRAM-ID MISSING OR MISPLACED. IF PROGRAM-ID DOES NOT IMMEDIATELY FOLLOW IDENTIFICATION DIVISION, IT WILL BE IGNORED.</td>
</tr>
<tr>
<td>22</td>
<td>IKF1087I-W</td>
</tr>
<tr>
<td></td>
<td>SOURCE 4 SHOULD NOT BEGIN A-MARGIN.</td>
</tr>
<tr>
<td>22</td>
<td>IKF1004I-E</td>
</tr>
<tr>
<td></td>
<td>INVALID WORD SOURCE 4 SKIPPING TO NEXT RECOGNIZABLE WORD.</td>
</tr>
<tr>
<td>27</td>
<td>IKF1087I-W</td>
</tr>
<tr>
<td></td>
<td>FILE 4 SHOULD NOT BEGIN A-MARGIN.</td>
</tr>
<tr>
<td>27</td>
<td>IKF1004I-E</td>
</tr>
<tr>
<td></td>
<td>INVALID WORD FILE 4 SKIPPING TO NEXT RECOGNIZABLE WORD.</td>
</tr>
<tr>
<td>28</td>
<td>IKF1003I-W</td>
</tr>
<tr>
<td></td>
<td>FILE-CONTROL PARAGRAPH NAME MISSING. ASSUMED PRESENT.</td>
</tr>
<tr>
<td>40</td>
<td>IKF1043I-W</td>
</tr>
<tr>
<td></td>
<td>END OF SENTENCE SHOULD PRECEDE 05 05 ASSUMED PRESENT.</td>
</tr>
<tr>
<td>75</td>
<td>IKF1004I-E</td>
</tr>
<tr>
<td></td>
<td>INVALID WORD VALUE 4 SKIPPING TO NEXT RECOGNIZABLE WORD.</td>
</tr>
<tr>
<td>94</td>
<td>IKF1043I-W</td>
</tr>
<tr>
<td></td>
<td>END OF SENTENCE SHOULD PRECEDE 05 05 ASSUMED PRESENT.</td>
</tr>
<tr>
<td>101</td>
<td>IKF1081I-W</td>
</tr>
<tr>
<td></td>
<td>PERIOD NOT FOLLOWED BY SPACE. ASSUME END OF SENTENCE.</td>
</tr>
<tr>
<td>101</td>
<td>IKF1080I-W</td>
</tr>
<tr>
<td></td>
<td>PERIOD PRECEEDED BY SPACE. ASSUME END OF SENTENCE.</td>
</tr>
<tr>
<td>122</td>
<td>IKF3001I-E</td>
</tr>
<tr>
<td></td>
<td>WS-TOTAL-DEPENDENTS NOT DEFINED. DISCARDED.</td>
</tr>
<tr>
<td>122</td>
<td>IKF3001I-E</td>
</tr>
<tr>
<td></td>
<td>WS-TL-NUMBER-OF-DEPENDENTS NOT DEFINED.</td>
</tr>
<tr>
<td>134</td>
<td>IKF4019I-E</td>
</tr>
<tr>
<td></td>
<td>DNM=1-427 (AN) MAY NOT BE USED AS ARITHMETIC OPERAND IN ADD STATEMENT. ARBITRARILY SUBSTITUTING TALLY.</td>
</tr>
<tr>
<td>134</td>
<td>IKF4019I-E</td>
</tr>
<tr>
<td></td>
<td>DNM=2-81 (NE) MAY NOT BE USED AS ARITHMETIC OPERAND IN ADD STATEMENT. ARBITRARILY SUBSTITUTING TALLY.</td>
</tr>
<tr>
<td>141</td>
<td>IKF3001I-E</td>
</tr>
<tr>
<td></td>
<td>WS-DL-NAME NOT DEFINED. DISCARDED.</td>
</tr>
<tr>
<td>144</td>
<td>IKF4019I-E</td>
</tr>
<tr>
<td></td>
<td>DNM=1-427 (AN) MAY NOT BE USED AS ARITHMETIC OPERAND IN ADD STATEMENT. ARBITRARILY SUBSTITUTING TALLY.</td>
</tr>
<tr>
<td>144</td>
<td>IKF5011I-W</td>
</tr>
<tr>
<td></td>
<td>AN INTERMEDIATE RESULT OR A SENDING FIELD MIGHT HAVE ITS HIGH ORDER DIGIT POSITION TRUNCATED.</td>
</tr>
<tr>
<td>147</td>
<td>IKF4019I-E</td>
</tr>
<tr>
<td></td>
<td>DNM=2-49 (NE) MAY NOT BE USED AS ARITHMETIC OPERAND IN ADD STATEMENT. ARBITRARILY SUBSTITUTING TALLY.</td>
</tr>
</tbody>
</table>
IDENTIFICATION DIVISION.

PROGRAM-ID. COBSYN1.

AUTHOR. SP6 DW CROSS, SOFTWARE.

INSTALLATION. CSD, USAIA, FT HARRISON, IN 46216.

DATE-WRITTEN. MAR 3 1978.


SECURITY. UNCLASSIFIED.

REMARKS. THIS PROGRAM LISTS ALL ORDERS RECEIVED EACH DAY. IT LISTS THE PART NUMBER, CUSTOMER NAME, AND CUSTOMER ADDRESS.

WS-ORDER-EOF-SW IS USED IN THE PROCEDURE DIVISION TO SIGNIFY WHEN THE END OF FILE CONDITION HAS BEEN REACHED.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-360-H40.

OBJECT-COMPUTER. IBM-360-H40.

SPECIAL- NAMES.

COL IS TOP-OF-PAGE.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT IM-ORDER-FILE.

ASSIGN TO UT-S-SYS006.

SELECT OR-ORDERS-LISTING.

ASSIGN TO UT-S-SYS005.

DATA DIVISION.

FILE SECTION.

FD IM-ORDER-FILE.

BLOCK CONTAINS 1 RECORDS.

RECORD CONTAINS 80 CHARACTERS.

LABEL RECORD IS STANDARD.

DATA RECORD IS IM01-ORDER-RCD.

01 IM01-ORDER-RCD.

05 IM01-PART-NO PIC 9(4).

05 IM01-CUSTOMER-NAME PIC X(18).

05 IM01-CUSTOMER-ADDRESS PIC X(18).

05 IM01-UNIT-OF-ISSUE PIC X.

05 IM01-ORDERED PIC 9(9).

05 IM01-PRICE-ITEM PIC 9(3)V99.

05 IM01-CUSTOMER-CODE PIC X.

05 FILLER PIC X(3).

FD OR-ORDERS-LISTING.

BLOCK CONTAINS 1 RECORDS.

RECORD CONTAINS 133 CHARACTERS.

LABEL RECORD IS STANDARD.

DATA RECORD IS OR01-ORDER-LINE.

01 OR01-ORDER-LINE PIC X(133).

WORKING-STORAGE SECTION.

77 WS-ORDER-EOF-SW VALUE 'OFF'.

01 WS-HEADER-LINE PIC X(33).

05 FILLER VALUE SPACES.
00055 05 FILLER VALUE 'PART NO'. PIC X(7).
00056 05 FILLER VALUE SPACES. PIC X(11).
00057 05 FILLER VALUE 'CUSTOMER NAME'. PIC X(13).
00058 05 FILLER VALUE SPACES. PIC X(12).
00059 05 FILLER VALUE 'CUSTOMER ADDRESS'. PIC X(16).
00060 05 FILLER VALUE SPACES. PIC X(41).
00061 05 FILLER VALUE SPACES. PIC X(13).
00062 05 FILLER VALUE SPACES. PIC X(35).
00063 05 WS-DL-PART-NO PIC 9(4).
00064 05 FILLER PIC (10) X(10).
00065 05 WS-DL-CUSTOMER-NAME PIC X(18).
00066 05 WS-DL-CUSTOMER-ADDRESS PIC X(40).
00067 05 FILLER VALUE SPACES. PIC X(35).
00068 05 FILLER VALUE SPACES. PIC X(35).
00069 05 WS-DL-PART-NO PIC 9(4).
00070 05 FILLER PIC (10) X(10).
00071 05 WS-DL-CUSTOMER-NAME PIC X(18).
00072 05 WS-DL-CUSTOMER-ADDRESS PIC X(40).
00073 05 FILLER VALUE SPACES. PIC X(35).
00074 05 FILLER VALUE SPACES. PIC X(35).
00075 05 WS-DL-CUSTOMER-NAME PIC X(18).
00076 05 FILLER VALUE SPACES. PIC X(35).
00077 05 FILLER VALUE SPACES. PIC X(35).
00078 05 WS-DL-CUSTOMER-ADDRESS PIC X(40).
00079 05 PROCEDURE DIVISION.
00080 0010-DRIVER.
00081 OPEN INPUT IM-ORDER-FILE
00082 OUTPUT OR-ORDERS-LISTING
00083 WRITE OR01-ORDER-LINE FROM WS-HEADING-LINE
00084 AFTER ADVANCING TOP-OF-PAGE
00085 PERFORM 0020-READ-LIST-RTN THRU 0020-EXIT
00086 UNTIL WS-ORDER-EOF-SW EQUAL 'ON'.
00087 CLOSE IM-ORDER-FILE
00088 OR-ORDERS-LISTING.
00089 0010-EXIT.
00090 STOP RUN.
00091 0020-READ-LIST-RTN.
00092 READ IM-ORDER-FILE
00093 AT END
00094 MOVE 'ON' TO WS-ORDER-EOF-SW
00095 GO TO 0020-EXIT.
00096 MOVE IM01-PART-NO TO WS-DL-PART-NO
00097 MOVE IM01-CUSTOMER-NAME TO WS-DL-CUSTOMER-NAME
00098 MOVE IM01-CUSTOMER-ADDRESS TO WS-DL-CUSTOMER-ADDRESS
00099 WRITE OR01-ORDER-LINE FROM WS-DETAIL-LINE
00100 AFTER ADVANCING 2 LINES.
00101 0020-EXIT.
00102 EXIT.
PROGRAM-ID. COBSYN2.
IDENTIFICATION DIVISION.

AUTHOR. SP6 OW GROSS.

INSTALLATION. CSF, USAIA, FT HARRISON, IN 46216.

DATE WRITTEN. MAR 3 1978.
SECURITY. UNCLASSIFIED.

*REMARKS. THIS PROGRAM LISTS ALL PERSONNEL IN THE
* BATTALION WITH OVER 75 MONTHS SERVICE.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-360-H40.
OBJECT-COMPUTER. IBM-360-H40.
SPECIAL- NAMES.

FILE-CONTROL.
SELECT IM-PERSONNEL-MASTER-FILE
ASSIGN TO UT-SYSC06.
SELECT OR-PRINTER
ASSIGN TO UT-SYSC05.

DATA DIVISION.
FILE SECTION.
FD IM-PERSONNEL-MASTER-FILE
BLOCK CONTAINS 1 RECORDS,
RECORD CONTAINS 80 CHARACTERS,
LABEL RECORD IS STANDARD,
DATA RECORD IS IM01-PERSONNEL-MASTER-RCD.

01 IM01-PERSONNEL-MASTER-RCD.
05 IM01-BATTALION PIC XX.
05 IM01-COMPANY PIC X.
05 IM01-SSAN PIC 9(9).
05 IM01-NAME PIC X(18).
05 IM01-PAY-GRADE PIC 17 XX.
05 IM01-RANK PIC X(3).
05 IM01-ETS PIC X(6).
05 IM01-DATE-OF-ENTRY PIC X(6).
05 IM01-MONTHS-OF-SERVICE PIC X(3).
05 IM01-NUMBER-OF-DEPENDENTS PIC 99.
05 IM01-PREVIOUS-ASSIGN-INFO PIC X(128).

FD OR-PRINTER,
BLOCK CONTAINS 1 RECORDS,
RECORD CONTAINS 133 CHARACTERS,
LABEL RECORD IS STANDARD,
DATA RECORD IS OR01-PRINTER-RCD.

01 OR01-PRINTER-RCD.
77 WS-MASTER-FILE-EOF-SWITCH PIC XXX VALUE 'OFF'.

77 WS-HEADING-LINE.
05 FILLER PIC X(5.) VALUE SPACES.
05 FILLER PIC X(5.) VALUE SPACES.

IA-01-03-03

336
PROCEDURE DIVISION.

0010 - DRIVER.
OPEN INPUT IM-PERSONNEL-MASTER-FILE
OUTPUT OR-PRINTER.
WRITE CROI-PRINTER-RCO FROM WS-HEADING-LINE
AFTER ADVANCING TOP-OF-PAGE.
PERFORM 0020-READ-LIST-RTN THRU 0020-EXIT 0020-READ-LIST-RTN
UNTIL WS-MASTER-FILE-EOF-SWITCH EQUAL 'ON'.
CLOSE IM-PERSONNEL-MASTER-FILE OR-PRINTER.
0010-EXIT.
STOP RUN.
0010-EXIT.
0020-READ-LIST-RTN.
READ IM-PERSONNEL-MASTER-FILE
AT END
MOVE 'ON' TO WS-MASTER-FILE-EOF-SWITCH
GO TO 0020-EXIT.
0020-EXIT.
0000-EXIT.
IDENTIFICATION DIVISION.
PROGRAM-ID. COBSYN3.
AUTHOR. SP6 DW GROSS.
INSTALLATION. CSU, USAF, FT HARRISON, IN 46216.
DATE-COMPiled. JUN 21, 1978

*REMARKS.*

THIS PROGRAM PRODUCES THE DEPENDENT INFORMATION
REPORT. IT LISTS ALL PERSONNEL IN THE BATTALION
WITH 5 OR MORE DEPENDENTS BY NAME, RANK, COMPANY,
AND SIZE OF FAMILY.

WS-MASTER-FILE-EOF-SWITCH IS USED TO SIGNIFY
WHEN THE END OF FILE CONDITION HAS BEEN
ENCOUNTERED.

WS-TOTAL-DEPENDENTS IS USED TO ACCUMULATE THE
DEPENDENTS FOR THE PURPOSE OF PRINTING IT IN
THE TOTAL LINE.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

SOURCE COMPUTER. IBM-360-64.
OBJECT COMPUTER. IBM-360-64.

SPECIAL-NAMES.
CDI IS TOP-OF-PAGE.

INPUT-OUTPUT SECTION.

FILE CONTROL.

SELECT IM-PERSONNEL-MASTER-FILE
ASSIGN TO UT-S-SYS006.
SELECT OR-PRINTER
ASSIGN TO UT-S-SYS005.

DATA DIVISION.
FILE SECTION.

FD IM-PERSONNEL-MASTER-FILE;
BLOCK CONTAINS 1 RECORDS;
RECORD CONTAINS 80 CHARACTERS;
LABEL RECORD IS STANDARD;
DATA RECORD IS IMO1-PERSONNEL-MASTER-RCD.
01 IMO1-PERSONNEL-MASTER-RCD
05 IMO1-BATTALION
05 IMO1-COMPANY
05 IMO1-SSAN
05 IMO1-NAME
05 IMO1-PAY-GRADE
05 IMO1-RANK
05 IMO1-ETS
05 IMO1-DATE-OF-ENTRY
05 IMO1-MONTHS-OF-SERVICE
05 IMO1-NUMBER-OF-DEPENDENTS
05 IMO1-PREVIOUS-ASSIGN-INFO

FD OR-PRINTER;
BLOCK CONTAINS 1 RECORDS;
RECORD CONTAINS 133 CHARACTERS;
LABEL RECORD IS STANDARD.
DATA RECORD IS URO1-PRINTER-RCO.
01 URO1-PRINTER-RCO
WORKING-STORAGE SECTION.
77 WS-LINE-COUNTER
   VALUE ZEROS.
77 WS-TOTAL-DEPENDENTS
   VALUE ZEROS.
77 WS-MASTER-FILE-EOF-SWITCH
   VALUE 'OFF'.
01 WS-HEADING-LINE-1.
   05 FILLER
      VALUE SPACES.
01 WS-HEADING-LINE-2.
   05 FILLER
      VALUE SPACES.
   05 FILLER
      VALUE 'DEPENDENT INFORMATION REPORT'.
   05 FILLER
      VALUE SPACES.
01 WS-DETAIL-LINE.
   05 FILLER
      VALUE SPACES.
   05 FILLER
      VALUE SPACES.
   05 FILLER
      VALUE 'TOTAL NUMBER OF DEPENDENTS WITHIN THE BATTALION IS'.
   05 FILLER
      VALUE SPACES.
00112 05 WS-TL-NUMBER-OF-DEPENDENTS  PIC 2(3).
00113 05 FILLER VALUE SPACES.
00114      PROCEDURE DIVISION.
00115 0010-DRIVER.
00116      OPEN INPUT IM-PERSONNEL-MASTER-FILE
00117      OUTPUT OR-PRINTER.
00118      PERFORM 0030-HEADING-RTN THRU 0030-EXIT.
00119      PERFORM 0020-READ-WRITE-RTN THRU 0020-EXIT
00120      UNTIL WS-MASTER-FILE-EOF-SWITCH EQUAL 'ON'.
00121      MOVE WS-TOTAL-DEPENDENTS TO WS-TL-NUMBER-OF-DEPENDENTS.
00122      WRITE OR01-PRINTER-REC FROM WS-TOTAL-LINE
00123      AFTER ADVANCING 3 LINES.
00124      CLOSE IM-PERSONNEL-MASTER-FILE,
00125      OR-PRINTER.
00126 0010-EXIT.
00127      0020-READ-WRITE-RTN.
00128
00129      REAC IM-PERSONNEL-MASTER-FILE
00130      AT END
00131      MOVE 'ON' TO WS-MASTER-FILE-EOF-SWITCH
00132      GO TO 0020-EXIT.
00133      ADC IMO1-NUMBER-OF-DEPENDENTS TO WS-TOTAL-DEPENDENTS.
00134      IF IMO1-NUMBER-OF-DEPENDENTS LESS THAN 4
00135      THEN
00136      GO TO 0020-EXIT.
00137      IF WS-LINE-COUNTER GREATER THAN 15
00138      THEN
00139      PERFORM 0030-HEADING-RTN THRU 0030-EXIT.
00140      MOVE IMO1-NAME TO WS-DL-NAME.
00141      MOVE IMO1-RANK TO WS-DL-RANK.
00142      MOVE IMO1-COMPANY TO WS-DL-COMPANY.
00143      ADD 1, IMO1-NUMBER-OF-DEPENDENTS GIVING WS-DL-FAMILY-SIZE.
00144      WRITE OR01-PRINTER-REC FROM WS-DETAIL-LINE
00145      AFTER ADVANCING 2 LINES.
00146      ADD 1 TO WS-LINE-COUNTER.
00147      0020-EXIT.
00148      EXIT.
00149      0030-HEADING-RTN.
00150      MOVE 0 TO WS-LINE-COUNTER.
00151      WRITE OR01-PRINTER-REC FROM WS-HEADING-LINE-1
00152      AFTER ADVANCING TOP-OF-PAGE.
00153      WRITE OR01-PRINTER-REC FROM WS-HEADING-LINE-2
00154      AFTER ADVANCING 3 LINES.
00155 0030-EXIT.
00156 0030-EXIT.
00157 0030-EXIT.
Kalamazoo County State Bank
SCHOOLCRAFT - MICHIGAN

PAY TO THE ORDER OF

PLEASE BE SURE TO DEDUCT ANY PER CHECK CHARGES OR SERVICE CHARGES THAT MAY APPLY TO YOUR ACCOUNT

CHECK NO. DATE
119 AUG 12

CHECKS ISSUED TO OR DESCRIPTION OF DEPOSIT
AMOUNT OF CHECK
CASH $10.00

AMOUNT OF DEPOSIT
$10.00
$946.23

INDIANAPOLIS POWER & LIGHT COMPANY
P.O. BOX 19958
INDIANAPOLIS, INDIANA 46206

KEEP THIS PART

FA 809 908 5568 211 643 2345
TX INDIANA SALES TAX 109

BILLING DATE 5-16-7 MRT 2345
LATE PAYMENT CHARGE ADDED AFTER 5-24-7 GROSS DUE 2948

AUGUST, 197

S M T W T F S
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

IA-01-01-16
CHECK PROBLEM HIERARCHY CHART

1000
CONTROL

2010
GET-READY

2020
WRITE-CHECK

2030
UPDATE-LEDGER

2040
FINISH-UP
<table>
<thead>
<tr>
<th>INPUT</th>
<th>PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNPAID-BILL</td>
<td>PERFORM 2010-GET-READY.</td>
<td>PAID-BILL</td>
</tr>
<tr>
<td></td>
<td>PERFORM 2020-WRITE-CHECK.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORM 2030-UPDATE-LEDGER.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORM 2040-FINISH-UP.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STOP.</td>
<td></td>
</tr>
</tbody>
</table>

"FOR INSTRUCTIONAL PURPOSES ONLY"
<table>
<thead>
<tr>
<th>INPUT</th>
<th>PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. UNPAID-BILL</td>
<td>OPEN UNPAID-BILL.</td>
<td>1. OPENED UNPAID-BILL.</td>
</tr>
<tr>
<td>2. CHECK-BOOK</td>
<td>OPEN CHECK-BOOK.</td>
<td>2. OPENED CHECK-BOOK</td>
</tr>
<tr>
<td>3. CALANDER</td>
<td>EXTRACT TODAY'S-DATE.</td>
<td>2.1 BLANK-CHECK</td>
</tr>
<tr>
<td></td>
<td>EXIT.</td>
<td>2.2 CHECK-LEDGER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. TODAY'S-DATE</td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>----------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>OPENED CHECK-BOOK</td>
<td>ADD 1 TO PREVIOUS-CK-NO.</td>
<td>WRITTEN-CHECK</td>
</tr>
<tr>
<td>BLANK-CHECK</td>
<td>MOVE PREVIOUS-CHK-NO TO CHECK-NO.</td>
<td>CHECK-NO</td>
</tr>
<tr>
<td>CHECK-LEDGER</td>
<td>MOVE BILL-PAYEE TO CHECK-PAYEE.</td>
<td>CHECK-DATE</td>
</tr>
<tr>
<td>PREVIOUS-CHK-NO</td>
<td>MOVE BILL-AMOUNT TO CHECK-AMT-NUMERIC.</td>
<td>CHECK-PAYEE</td>
</tr>
<tr>
<td>TODAYS-DATE</td>
<td>MOVE BILL-AMOUNT TO CHECK-AMT-ALPHA.</td>
<td>CHECK-AMOUNT</td>
</tr>
<tr>
<td></td>
<td>MOVE TODAYS-DATE TO CHECK-DATE</td>
<td></td>
</tr>
<tr>
<td>OPENED UNPAID-BILL</td>
<td>SIGN CHECK.</td>
<td>CHK-AMT-</td>
</tr>
<tr>
<td>BILL-PAYEE</td>
<td>EXIT.</td>
<td>CHK-AMT-ALPHA</td>
</tr>
<tr>
<td>BILL-AMOUNT</td>
<td></td>
<td>SIGNATURE</td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>1.</td>
<td>CHECK</td>
<td>1. UPDATED-LEDGER</td>
</tr>
<tr>
<td>1.1</td>
<td>CHECK-NO</td>
<td>1.1 LEDGER-CHK-NO</td>
</tr>
<tr>
<td>1.2</td>
<td>CHECK-DATE</td>
<td>1.2 LEDGER-DATE</td>
</tr>
<tr>
<td>1.3</td>
<td>CHECK-PAYEE</td>
<td>1.3 LEDGER-PAYEE</td>
</tr>
<tr>
<td></td>
<td>MOVE CHECK-NO TO LEDGER-CHK-NO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE CHECK-DATE TO LEDGER-DATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE CHECK-PAYEE TO LEDGER-PAYEE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE CHECK-AMT-NUMERIC TO LEDGER-AMT-OF-CHK</td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>CHECK-AMT-NUMERIC</td>
<td>1.4 LEDGER-AMT-OF-CHK</td>
</tr>
<tr>
<td></td>
<td>SUBTRACT CHK-AMT-NUMERIC FROM LEDGER-OLD-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NUMERIC BALANCE</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>CHECK-LEDGER</td>
<td>1.5 LEDGER-NEW-BALANCE</td>
</tr>
<tr>
<td>2.1</td>
<td>MOVE LEDGER-OLD-BALANCE TO LEDGER-NEW-BALANCE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LEDGER-OLD-BALANCE EXIT</td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>1. OPENED CHECK-BOOK</td>
<td>REMOVE WRITTEN CHECK FROM CHECK BOOK.</td>
<td>1. WRITTEN-CHECK</td>
</tr>
<tr>
<td>2. OPENED UNPAID-BILL</td>
<td>MARK BILL PAID.</td>
<td>2. CLOSED-CHECK-BOOK</td>
</tr>
<tr>
<td></td>
<td>CLOSED CHECK-BOOK.</td>
<td>2.1 UPDATED-LEDGER</td>
</tr>
<tr>
<td></td>
<td>EXIT.</td>
<td>3. PAID-BILL</td>
</tr>
</tbody>
</table>
CHECK PROBLEM MODIFICATION HIERARCHY CHART

1000
CONTROL

2010
GET-READY

2020
PROCESS

2040
FINISH-UP

3010
WRITE-CHECK

3020
UPDATE-LEDGER
<table>
<thead>
<tr>
<th>INPUT</th>
<th>PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. UNPAID-BILL(S)</td>
<td>PERFORM 2010-GET-READY.</td>
<td>1. PAID-BILL(S)</td>
</tr>
<tr>
<td></td>
<td>PERFORM 2020-PROCESS UNTIL OUT OF BILLS.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORM 2040-FINISH-UP.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STOP.</td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>UNPAID-BILL</td>
<td>PERFORM 3010-WRITE-CHECK.</td>
<td>PAID-BILL</td>
</tr>
<tr>
<td></td>
<td>PERFORM 3020-UPDATE-LEDGER.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXIT.</td>
<td></td>
</tr>
</tbody>
</table>
CHECK PROBLEM MODIFICATION

Now that you have solved the problem of writing a check and updating the check ledger to pay a bill, you will modify your solution to follow this process until you have paid all the bills.

You will assume that there is more than enough in the account to cover any and all bills that you must pay.

To accomplish this, you must redo your HIPO, and modify or create new IPOs.
PROBLEM SOLVING PE #1 HIERARCHY CHART

1000
CONTROL

2010
GET-READY

2020
PROCESS

2030
FINISH-UP

3010
MOVE

3020
WRITE-CONTROL

3030
READ

4010
WRITE-HEADER

4020
WRITE-DETAIL
<table>
<thead>
<tr>
<th>INPUT</th>
<th>PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. PERSONNEL-MASTER-FILE</td>
<td>PERFORM 2010-GET-READY.</td>
<td>1. PRINTED-REPORT</td>
</tr>
<tr>
<td></td>
<td>PERFORM 2020-PROCESS UNTIL EOF-SW = 'ON'.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORM 2030-FINISH-UP.</td>
<td>EOF-SW = 'OFF'</td>
</tr>
<tr>
<td>EOF-SW = 'OFF' OR 'ON'</td>
<td>STOP RUN.</td>
<td>EOF-SW = 'OFF'</td>
</tr>
</tbody>
</table>

"FOR INSTRUCTIONAL PURPOSES ONLY"
<table>
<thead>
<tr>
<th>INPUT</th>
<th>PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. CLOSED PERSONNEL-</td>
<td>OPEN INPUT PERSONNEL-MASTER-FILE.</td>
<td>1. OPENED PERSONNEL-</td>
</tr>
<tr>
<td>MASTER-FILE</td>
<td>OPEN OUTPUT PRINT-FILE.</td>
<td>MASTER-FILE</td>
</tr>
<tr>
<td>2. CLOSED PRINT-FILE</td>
<td>MOVE 'OFF' TO EOF-SW.</td>
<td>2. OPENED PRINT-FILE</td>
</tr>
<tr>
<td></td>
<td>MOVE 'OFF' TO HEADER-SW.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORM 3030-READ.</td>
<td></td>
</tr>
<tr>
<td>EOF-SW</td>
<td>EXIT.</td>
<td>EOF-SW = 'OFF'</td>
</tr>
<tr>
<td>HEADER-SW</td>
<td></td>
<td>HEADER-SW = 'OFF'</td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>PERFORM 3010-MOVE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORM 3020-WRITE-CONTROL.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORM 3030-READ.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXIT.</td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>1. OPENED PERSONNEL-</td>
<td>CLOSE PERSONNEL-MASTER-FILE.</td>
<td>1. CLOSED PERSONNEL-</td>
</tr>
<tr>
<td>MASTER-FILE</td>
<td></td>
<td>MASTER-FILE</td>
</tr>
<tr>
<td>2. OPENED PRINT-FILE</td>
<td>CLOSE PRINT-FILE.</td>
<td>2. CLOSED PRINT-FILE</td>
</tr>
<tr>
<td></td>
<td>EXIT.</td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>PERSONNEL-MASTER-</td>
<td>MOVE PMR-NAME TO DL-NAME.</td>
<td>1. FORMATTED DETAIL-LINE</td>
</tr>
<tr>
<td>1.1 PMR-NAME</td>
<td>MOVE PMR-RANK TO DL-RANK.</td>
<td>1.1 DL-RANK</td>
</tr>
<tr>
<td>1.2 PMR-RANK</td>
<td>EXIT.</td>
<td>1.2 DL-NAME</td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>IF HEADER-SW = 'OFF'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THEN PERFORM 4010-WRITE-HEADER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE 'ON' TO HEADER-SW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELSE NEXT SENTENCE,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERFORM 4020-WRITE-DETAIL.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HEADER-SW = 'OFF' OR 'ON' EXIT.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HEADER-SW = 'OFF' OR 'ON'.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"FOR INSTRUCTIONAL PURPOSES ONLY"
<table>
<thead>
<tr>
<th>Input</th>
<th>Process</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. PERSONNEL-MASTER-</td>
<td>READ PERSONNEL-MASTER-FILE</td>
<td>1. PERSONNEL-MASTER-</td>
</tr>
<tr>
<td></td>
<td>AT END</td>
<td>RECORD</td>
</tr>
<tr>
<td></td>
<td>MOVE 'ON' TO EOF-SW.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXIT.</td>
<td></td>
</tr>
<tr>
<td>EOF-SW = 'OFF' OR 'ON'</td>
<td></td>
<td>EOF-SW = 'OFF' OR 'ON'</td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>WRITE HEADER-RECORD AFTER ADVANCING TO TOP OF PAGE.</td>
<td>1. HEADER-RECORD</td>
</tr>
<tr>
<td></td>
<td>EXIT.</td>
<td></td>
</tr>
<tr>
<td>1. HEADER-RECORD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1 'RANK'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2 'NAME'</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>INPUT</strong></td>
<td><strong>PROCESS</strong></td>
<td><strong>OUTPUT</strong></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>1. FORMATED-DETAIL-</td>
<td>WRITE DETAIL-RECORD AFTER ADVANCING 2 LINES.</td>
<td>1. DETAIL-RECORD</td>
</tr>
<tr>
<td></td>
<td>EXIT.</td>
<td></td>
</tr>
</tbody>
</table>

"FOR INSTRUCTIONAL PURPOSES ONLY"
PROBLEM SOLVING PE #2

I. The records section here at Fort Harrison is not satisfied with the report from PE #1. They have asked for the following modifications.

Count the lines on each page and print only twenty (20) detail lines on each page.

Format the output report as follows;

FORT HARRISON PERSONNEL LISTING

<table>
<thead>
<tr>
<th>RANK</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXX</td>
<td>XXXXXXXXXXXXXXXXXXXXX</td>
</tr>
<tr>
<td>XXX</td>
<td>XXXXXXXXXXXXXXXXXXXXX</td>
</tr>
</tbody>
</table>

Print the heading lines at the top of each new page.
<table>
<thead>
<tr>
<th>INPUT</th>
<th>PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. CLOSED PERSONNEL-MASTER-FILE</td>
<td>OPEN INPUT PERSONNEL-MASTER-FILE.</td>
<td>1. OPENED PERSONNEL-MASTER-FILE</td>
</tr>
<tr>
<td></td>
<td>OPEN OUTPUT PRINT-FILE.</td>
<td></td>
</tr>
<tr>
<td>2. CLOSED PRINT-FILE</td>
<td>MOVE 'OFF' TO EOF-SW.</td>
<td>2. OPENED PRINT-FILE</td>
</tr>
<tr>
<td></td>
<td>MOVE 99 TO LINE-CNTR.</td>
<td></td>
</tr>
<tr>
<td>EOF-SW</td>
<td>PERFORM 3030-READ.</td>
<td>EOF-SW = 'OFF'</td>
</tr>
<tr>
<td>LINE-CNTR</td>
<td>EXIT.</td>
<td>LINE-CNTR = 99</td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>IF LINE-CNTR IS GREATER THAN 19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THEN PERFORM 4010-WRITE-HEADER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE 0 TO LINE-CNTR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELSE NEXT SENTENCE.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERFORM 4020-WRITE-DETAIL.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINE-CNTR = (N) ADD 1 TO LINE-CNTR.</td>
<td>LINE-CNTR = (N+1) OR 0</td>
<td></td>
</tr>
<tr>
<td>EXIT.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"FOR INSTRUCTIONAL PURPOSES ONLY"
<table>
<thead>
<tr>
<th>INPUT</th>
<th>PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WRITE HEADER-RECORD-1 AFTER ADVANCING TO TOP OF PAGE.</td>
<td>1. HEADER-RECORD-1</td>
</tr>
<tr>
<td></td>
<td>WRITE HEADER-RECORD AFTER ADVANCING 2 LINES.</td>
<td>2. HEADER-RECORD</td>
</tr>
<tr>
<td>1. HEADER-RECORD-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1 'FORT HARRISON PERSONNEL LISTING'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. HEADER-RECORD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1 'RANK'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2 'NAME'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PROBLEM SOLVING PE #3

I INTRODUCTION.

The Battalion Headquarters at Fort Harrison has asked for a listing of all personnel assigned to or stationed here for accountability purposes. They would also like a count of all records listed.

II INPUT/OUTPUT.

INPUT: Input will consist of a personnel Master file, which contains records on all personnel stationed at or assigned to the Battalion.

Personnel Master Records;
1. SOCIAL SECURITY ACCOUNT NUMBER
2. NAME
3. BATTALION
4. RANK
5. NUMBER OF DEPENDENTS

OUTPUT: Output will consist of a printed listing with a heading at the top of each page. Print a detail line for each record read, and a total line showing the number of records processed. Write 20 detail lines per page (double spaced) then start at the top of the next page with the header. The total line will be printed only once at the end of the report.

Format the output report as follows;

<table>
<thead>
<tr>
<th>SSN</th>
<th>NAME</th>
<th>RANK</th>
<th>NR DEPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>999999999</td>
<td>xxxxxxxxxxxxxxxxxxxxxxxx</td>
<td>xxx</td>
<td>99</td>
</tr>
<tr>
<td>999999999</td>
<td>xxxxxxxxxxxxxxxxxxxxx:xxxxx</td>
<td>xxx</td>
<td>99</td>
</tr>
<tr>
<td>999999999</td>
<td>xxxxxxxxxxxxxxxxxxxxxxx</td>
<td>xxx</td>
<td>99</td>
</tr>
</tbody>
</table>

TOTAL RECORDS PRINTED 999
PROBLEM SOLVING PE #3 HIERARCHY CHART

1000
CONTROL

2010
GET-READY

2020
COUNT-
PROCESS

2030
WRITE-
TOTAL

2040
FINISH-
UP.

3010
MOVE

3020
WRITE-
CONTROL

3030
READ

4010
WRITE-
HEADER

4020
WRITE-
DETAIL

3030
READ
<table>
<thead>
<tr>
<th>INPUT</th>
<th>PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>1. CLOSED PERSONNEL-MASTER-FILE</td>
<td>OPEN INPUT PERSONNEL-MASTER-FILE.</td>
<td>1. OPENED PERSONNEL-MASTER-FILE</td>
</tr>
<tr>
<td>2. CLOSED PRINT-FILE</td>
<td>OPEN OUTPUT PRINT-FILE</td>
<td>2. OPENED PRINT-FILE</td>
</tr>
<tr>
<td></td>
<td>MOVE 99 TO LINE-CNTR.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE 'OFF' TO EOF-SW.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE 0 TO PERS-CNTR.</td>
<td></td>
</tr>
<tr>
<td>LINE-CNTR</td>
<td>PERFORM 3030-READ.</td>
<td>LINE-CNTR = 99</td>
</tr>
<tr>
<td>EOF-SW</td>
<td>EXIT.</td>
<td>EOF-SW = 'OFF'</td>
</tr>
<tr>
<td>PERS-CNTR</td>
<td></td>
<td>PERS-CNTR = 0</td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>ADD 1 TO PERS-CNTR.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORM 3010-MOVE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORM 3020-WRITE-CONTROL.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORM 3030-READ.</td>
<td></td>
</tr>
<tr>
<td>PERS-CNTR = (N)</td>
<td>EXIT.</td>
<td>PERS-CNTR = (N+1)</td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td></td>
<td>MOVE PERS-CNTR TO TOTAL-PERS.</td>
<td>1. TOTAL-LINE</td>
</tr>
<tr>
<td></td>
<td>WRITE TOTAL-LINE AFTER ADVANCING 2 LINES.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXIT.</td>
<td></td>
</tr>
</tbody>
</table>

1. TOTAL-LINE

1.1 'TOTAL RECORDS PRINTED'

1.2 TOTAL-PERS

PERS-CNTR = (N)
<table>
<thead>
<tr>
<th>INPUT</th>
<th>PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. OPENED PERSONNEL-MASTER-FILE</td>
<td>CLOSE PERSONNEL-MASTER-FILE.</td>
<td>1. CLOSED PERSONNEL-MASTER-FILE</td>
</tr>
<tr>
<td>2. OPENED PRINT-FILE</td>
<td>EXIT.</td>
<td>2. CLOSED PRINT-FILE</td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>PERSONNEL-MASTER-RECORD</td>
<td>MOVE PMR-SSN TO DL-SSN.</td>
<td>1. FORMATED-DETAIL-LINE</td>
</tr>
<tr>
<td>1.1 PMR-SSN</td>
<td>MOVE PMR-RANK TO DL-RANK.</td>
<td>1.1 DL-SSN</td>
</tr>
<tr>
<td>1.2 PMR-NAME</td>
<td>MOVE PMR-NR-DEPS TO PMR-NR-DEPS.</td>
<td>1.2 DL-NAME</td>
</tr>
<tr>
<td>1.3 PMR-RANK</td>
<td>EXIT.</td>
<td>1.3 DL-RANK</td>
</tr>
<tr>
<td>1.4 PMR-NR-DEPS</td>
<td></td>
<td>1.4 DL-NR-DEPS</td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td>IF LINE-CNTR IS GREATER THAN 19 THEN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORM 4010-WRITE-HEADER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE 0 TO LINE-CNTR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ELSE NEXT SENTENCE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORM 4020-WRITE-DETAIL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADD 1 TO LINE-CNTR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LINE-CNTR = (N)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXIT.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LINE-CNTR = (N+1) OR 0</td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>PERSONNEL-MASTER-FILE</td>
<td>READ PERSONNEL-MASTER-FILE</td>
<td>PERSONNEL-MASTER-RECORD</td>
</tr>
<tr>
<td>AT-END</td>
<td>MOVE 'ON' TO EOF-SW.</td>
<td>EOF-SW = 'OFF' OR 'ON'</td>
</tr>
<tr>
<td>EOF-SW = 'OFF'</td>
<td>EXIT.</td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>WRITE HEADER-LINE AFTER ADVANCING TO TOP</td>
<td>1. HEADER-LINE OF PAGE</td>
<td></td>
</tr>
<tr>
<td>EXIT.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. HEADER-LINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1 'SSN'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2 'NAME'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3 'RANK'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4 'NR DEPS'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**For instructional purposes only**
<table>
<thead>
<tr>
<th>INPUT</th>
<th>PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMATTED DETAIL-LINE</td>
<td>WRITE DETAIL-RECORD AFTER ADVANCING 2</td>
<td>DETAIL-RECORD</td>
</tr>
<tr>
<td></td>
<td>LINES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXIT</td>
<td></td>
</tr>
</tbody>
</table>
I INTRODUCTION.

The Battalion Headquarters was pleased with the report produced from PE #3 but feel they could better use the report with the following modifications.

1. Print only those records of personnel above the rank of PV1.

2. Print an additional total line two lines below the one now being generated which contains a count of the number of PV1's on the Personnel Master File.

Format of second total line is as follows;

TOTAL NUMBER OF PV1 999
<table>
<thead>
<tr>
<th>INPUT</th>
<th>PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MOVE PERS-CNTR TO TOTAL-PERS.</td>
<td>1. TOTAL-LINE</td>
</tr>
<tr>
<td></td>
<td>WRITE TOTAL-LINE AFTER ADVANCING 2 LINES.</td>
<td>2. TOTAL-LINE-2</td>
</tr>
<tr>
<td>1. TOTAL-LINE</td>
<td>MOVE PV1-CNTR TO TOTAL-PV1.</td>
<td></td>
</tr>
<tr>
<td>1.1 TOTAL RECORDS</td>
<td>'TOTAL RECORDS LINES,' PRINTED'</td>
<td></td>
</tr>
<tr>
<td>1.2 TOTAL-PERS</td>
<td>'TOTAL NUMBER OF PV1'</td>
<td></td>
</tr>
<tr>
<td>2. TOTAL-LINE-2</td>
<td>'TOTAL NUMBER OF PV1'</td>
<td></td>
</tr>
<tr>
<td>2.1 TOTAL-PV1</td>
<td>PERS-CNTR = (N)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PV1-CNTR = (N)</td>
<td></td>
</tr>
</tbody>
</table>

"FOR INSTRUCTIONAL PURPOSES ONLY"
<table>
<thead>
<tr>
<th>INPUT</th>
<th>PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERSONNEL-MASTER-</td>
<td>IF PMR-RANK = 'PV1'</td>
<td></td>
</tr>
<tr>
<td>RECORD</td>
<td>THEN ADD 1 TO PV1-CNTR</td>
<td></td>
</tr>
<tr>
<td>PMR-RANK</td>
<td>ELSE ADD 1 TO PERS-CNTR</td>
<td></td>
</tr>
<tr>
<td>PERFORM 3010-MOVE</td>
<td>PERFORM 3020-WRITE-CONTROL.</td>
<td></td>
</tr>
<tr>
<td>PERFORM 3030-READ</td>
<td>PERS-CNTR = (N) OR (N+1)</td>
<td></td>
</tr>
<tr>
<td>PERS-CNTR = (N)</td>
<td>EXIT.</td>
<td></td>
</tr>
<tr>
<td>PV1-CNTR = (N)</td>
<td>PV1-CNTR = (N) OR (N+1)</td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>PROCESS</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>1. CLOSED PERSONNEL-</td>
<td>OPEN INPUT PERSONNEL-MASTER-FILE.</td>
<td>1. OPENED PERSONNEL-</td>
</tr>
<tr>
<td>MASTER-FILE</td>
<td></td>
<td>MASTER-FILE</td>
</tr>
<tr>
<td>2. CLOSED PRINT-FILE</td>
<td>OPEN OUTPUT PRINT-FILE.</td>
<td>2. OPENED PRINT-FILE</td>
</tr>
<tr>
<td>Line-CNTR</td>
<td>MOVE 99 TO LINE-CNTR.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE 'OFF' TO EOF-SW.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE 0 TO PERS-CNTR.</td>
<td></td>
</tr>
<tr>
<td>EOF-SW</td>
<td>MOVE 0 TO PV1-CNTR.</td>
<td></td>
</tr>
<tr>
<td>PERS-CNTR</td>
<td>EXIT.</td>
<td></td>
</tr>
<tr>
<td>PV1-CNTR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"FOR INSTRUCTIONAL PURPOSES ONLY"
SORT EXERCISES

1. Code the jobstream to sort a file. I/O specifications follow (Use 1 sort work area):

<table>
<thead>
<tr>
<th>Media:</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tape</td>
<td>80/1600</td>
<td>Tape</td>
</tr>
<tr>
<td></td>
<td>PERS-TX</td>
<td>80/1600</td>
</tr>
<tr>
<td>Rec Length:</td>
<td>500 records</td>
<td>SORTED-PERS-TX</td>
</tr>
<tr>
<td>File ID:</td>
<td>TP6</td>
<td>500 records</td>
</tr>
<tr>
<td>Volume of Data</td>
<td>pos 8-15 (character data)</td>
<td>TP1</td>
</tr>
<tr>
<td>Device:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control Field</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Assume that you have 3 sorted files in the format of problem 1. Code the jobstream to merge them.

3. Code the jobstream to sort the following files (Use 2 sort work extents):

<table>
<thead>
<tr>
<th>Media:</th>
<th>Input1</th>
<th>Input2</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tape</td>
<td>100/1000</td>
<td>Tape</td>
<td>Disk</td>
</tr>
<tr>
<td>Rec length/Blksise:</td>
<td>LOG-MSTR</td>
<td>100/1000</td>
<td>100/7000</td>
</tr>
<tr>
<td>File ID:</td>
<td>LOG-TEMP</td>
<td>LOG-TEMP</td>
<td>SORTED-LOG-MSTR</td>
</tr>
<tr>
<td>Volume of data:</td>
<td>20000 records</td>
<td>30000 records</td>
<td>50000 records</td>
</tr>
<tr>
<td>Device:</td>
<td>TP1</td>
<td>TP0</td>
<td>DK4</td>
</tr>
<tr>
<td>Volume Serial Nr:</td>
<td>N/A</td>
<td>N/A</td>
<td>LOG PAK</td>
</tr>
<tr>
<td>Retention:</td>
<td>N/A</td>
<td>N/A</td>
<td>3 months</td>
</tr>
<tr>
<td>Control fields:</td>
<td>8-20, char</td>
<td>29-30, char</td>
<td></td>
</tr>
<tr>
<td>MAJOR:</td>
<td>8-20, char</td>
<td>29-30, char</td>
<td></td>
</tr>
<tr>
<td>INT 1:</td>
<td>24, char</td>
<td>24, char</td>
<td></td>
</tr>
<tr>
<td>INT 2:</td>
<td>30-36, char</td>
<td>30-36, char</td>
<td></td>
</tr>
</tbody>
</table>
STRUCTURED FLOWCHARTING SYMBOLS

The flowcharting symbols defined in this handout are to be used in conjunction with the Verbs defined in the COBOL SOURCE DESIGN LANGUAGE handout. This handout does not contain all of the symbols that can be used in structured flowcharting but is limited to those symbols that will be used in the problem solving block of instruction.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS SYMBOL</td>
<td>The process symbol may be used to represent the ADD, CLOSE, OPEN, MOVE or NEXT SENTENCE. This symbol requires one entry and one exit.</td>
</tr>
<tr>
<td>PREDEFINED PROCESS SYMBOL</td>
<td>The predefined process symbol may be used only to represent a PERFORM. This symbol requires one entry and one exit.</td>
</tr>
<tr>
<td>DECISION</td>
<td>The decision symbol may be used only to represent the IF-THEN-ELSE. This symbol requires one entry and two exits, one THEN exit and one ELSE exit.</td>
</tr>
<tr>
<td>PUNCHED CARD</td>
<td>The punched card symbol may be used to represent a READ for an input punched card file or a WRITE of a punched card record. When used for a read it will have one entry and two exits. When used for a write it will have one entry and one exit.</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>USE</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>DOCUMENT</td>
<td>The document symbol may be used only to represent a WRITE to a printer. This symbol requires one entry and one exit.</td>
</tr>
<tr>
<td>MAGNETIC TAPE</td>
<td>The magnetic tape symbol is used in the same way as the punched card symbol except where the storage media is magnetic tape.</td>
</tr>
<tr>
<td>ONLINE STORAGE</td>
<td>The online storage symbol is used in the same way as the punched card symbol except where the storage media is a disk.</td>
</tr>
<tr>
<td>INPUT/OUTPUT</td>
<td>The input/output symbol is used in the same way as the punched card symbol except where the storage media is unknown.</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>The terminal symbol is used to show the beginning and ending point of each routine. It will contain the module number and name or the word EXIT or STOP RUN. This symbol requires either one entry or one exit but not both.</td>
</tr>
</tbody>
</table>
### SYMBOL

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON PAGE CONNECTOR</td>
<td>The on page connector symbol is used to connect two parts of the same routine which are drawn separately on the same sheet of paper.</td>
</tr>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td></td>
</tr>
<tr>
<td>OFF PAGE CONNECTOR</td>
<td>The off page connector symbol is used to connect two parts of the same routine which are drawn separately on separate sheets of paper.</td>
</tr>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>

When using the PUNCH CARD, MAGNETIC TAPE, ONLINE STORAGE or INPUT/OUTPUT symbol for a READ, you will always show the AT END associated with it as follows: (See page 4)

For an illustration of the use of an IF-THEN-ELSE and a compound IF-THEN-ELSE see page 5.
1. Code the jobstream to copy a deck of cards to tape. Columns 21 and 30 must be interchanged on the output tape, additionally columns 71-80 must be switched with columns 1-10. All other columns remain the same. The output tape must have 20 records per block and should be unloaded at EOJ.

2. Code the jobstream to list a file on disk. The file is on tracks 80-200 on pack DATA19. The records on disk are 100 bytes long and blocked 20 records per block. The listing should be single spaced. A 2314 disk will be used. The file-ID of the disk file is 'MOD WORKOUT'.

3. Write the jobstream to punch cards from a disk file. The file is called 'PAY RECORDS' and is on pack PAYPAC (omit extent info). The records are 120 bytes blocked by 20. The first 40 and last 40 bytes make up the 80 columns of output data. The first 1500 records should be bypassed.

4. Write the jobstream to print the cards punched in problem 3. All 80 bytes are to be printed with the output having 5 spaces between each group of 40 bytes (i.e.: 40 bytes 5 spaces, 40 bytes). The listing should be double spaced with page numbers.

5. Write the jobstream to load a tape file to disk. The input file-ID is 'A01ABC TRANS' and the record length is 100 blocked by 30. The output file is stored on pack DLOGO1 with extents and file-ID determined by the programmer. Output retention is one month.

6. Write the jobstream to copy a tape file. Input tape is 'PAY FILE' created on 6 Jan and is 90 bytes per record blocked by 10. Output should be blocked by 30 and be retained 3 months. Input file is generation 5, output is generation 6.
The PSNCO of your unit has brought you a deck of cards that he wants listed. The format for these cards is attached. The cards represent all enlisted personnel in the battalion. He does not know the current sequence of these cards, but he requires the output to be printed in sequence according to the print record layout below. Code the jobstream that will sort the cards and print them according to the customer's desires.

NOTES:

1. The SORT program will not read cards or print so two utility programs are required; one to transfer the card data to disk and another to print the sorted file.

2. Use 1 sort work file on disk.

3. Use ADAS JCL for all disk files (Volume serial no = SYSWRK).

<table>
<thead>
<tr>
<th>Field</th>
<th>Sequence</th>
<th>Card</th>
<th>Print</th>
</tr>
</thead>
<tbody>
<tr>
<td>Social Security Number</td>
<td>1-9</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Minor; A</td>
<td>11-30</td>
<td>21-40</td>
</tr>
<tr>
<td>MOS</td>
<td></td>
<td>32-38</td>
<td>42-48</td>
</tr>
<tr>
<td>Grade</td>
<td>Int; D</td>
<td>40-42</td>
<td>50-52</td>
</tr>
<tr>
<td>Company</td>
<td>Major; D</td>
<td>44</td>
<td>14</td>
</tr>
<tr>
<td>Date of Rank</td>
<td>46-51</td>
<td>56-61</td>
<td></td>
</tr>
</tbody>
</table>
DISK OPERATING SYSTEM
INTERFILE TRANSFER AND MANIPULATOR PROGRAM
FOR AID IN TESTING AND OPERATIONS
"DITTO"
# TABLE OF CONTENTS

I. Title Page 1

II. Table of Contents 2

III. Program Abstract 3

IV. Description of Functions 4

V. Control Card Operation 10

VI. Console Operating Instructions 12

VII. Multiprogramming Considerations 13

VIII. Table of Function Codes 15

IX. Operating Suggestions 17

X. Implementation 19

XI. Tape and Deck Key 20

XII. Sample Programs 21

Ditto is used throughout this documentation in lieu of the program title, "DOS Interfile Transfer and Manipulator Program For Aid In Testing and Operations."
PROGRAM ABSTRACT

DOS Interfile Transfer And Manipulator Program For Aid In Testing And Operations - (DITTO), is a generalized program which operates in a DOS environment. Thirty-three (33) functions are available to support Unit Record, Tape and 2311 and 2314 Disk. Functions available in addition to normal tape, disk and card functions are: Tape and Disk Record Scan, Disk and Tape Record Alteration, Disk ID Volume Number Change, Initialize Tape, Deblocking of Tape and Disk Records When Printing, and User Tape Error Handling.

Operating Characteristics Are:

1. The program is self-relocating to allow operating in any partition. Tape and Disk record size is limited only by the amount of core available at execution time.

2. Operations may be performed entirely from the console or function card statements may be job-streamed for remote programmer use.

3. All console communication is in terms of physical hardware I/O addresses, eliminating the need for knowledge of current logical assignments. Tape density and mode settings may be entered via the console.

4. DITTO functions in a DASD file protected environment without volume label statements. Printed output reflects the physical and logical addresses and file characteristics (density, etc.) of the input file.

5. Minimum requirements: System/360 DOS System with a 1403 Printer and a minimum 14K problem program area. The program is written in Assembler language.

DESCRIPTION OF FUNCTIONS

Disk Print

2311 or 2314 Disk to Printer in character and vertical format within specified limits. Disk records are "separated" into key and data format and printed with their associated cylinder, head, and record addresses. When a defective track is detected, the assigned alternate track with its respective data is printed. This provides a sequential listing of the disk file in "logical" sequence.

Two Disk to Printer formats are available: Disk Dump Unblocked; and Disk Dump with Reblocking. The Unblocked format lists the actual records as they exist on the file. The Reblock function allows the user to submit a logical record size. This size parameter should be equivalent to the actual logical record length only, and should not include separate key length. Imbedded keys should, however, be included. The Reblock function lists the record key and then deblocks the data portion of the record into logical lengths. This

CSD.SFW.DITTO
provides a more easily used listing of large physical record files.

The Reblock function is primarily intended for use on sequential or random files. Because of the different formats present on index sequential files, reblocking may not be meaningful for overflow areas or cylinder and track indices.

**Disk Record Scan**

Disk Record Scan provides the capability of scanning a disk file and recording the locations of all records which match a given scan argument. Three (3) scan argument types are available.

"Scan on Key" allows the user to scan for a match on any portion of the disk record key fields. A 1 to 35 position scan argument, and its associated starting position within the key field, are entered via the console. Beginning and ending disk limits are also entered. DITTO will scan the specified disk area and log all "hits" in the form of cylinder, head and record numbers on SYSLOG.

"Scan on Data" provides a similar capability for the data portion of disk records. A logical record length must be specified to allow internal de-blocking for scanning blocked files.

"Scan on EOF" will log all End of File record locations. Data entry for "EOF" is not required.

**Disk Record Load**

This function will alter the contents of the key and/or data portion of an existing 2311 or 2314 Disk Record. After the required disk unit and record addresses have been entered, the record will be retrieved and "separated" into key and data format and printed on SYSLST.

At this point, DITTO will request the number of bytes to be changed and the field type (Key or Data portion of the record) to be altered. After specifying these parameters, a starting position within the field will be requested. This starting position is based on using the first position of the key or data as position "0001". The user can verify this position by referencing the printout of the record and its associated scale on SYSLST. Data may be entered via the console in character (1 character per byte of change), or hexadecimal (2 characters per byte of change). Valid hexadecimal "characters" are 0-9 and A through F.

After all alterations to the record in core have been made, the altered record will again be printed on SYSLST and a message issued "ARE CHANGES COMPLETE - Y OR N." A response of "N" (NO) will allow the user to re-enter the
alter routine. A response of "Y" (YES) will rewrite the record onto disk.

Write Disk End-of-File Record

This function will retrieve any existing 2311 or 2314 Disk Record and re-
write the record as an end-of-file record (key and data length of zero).
The user must supply the cylinder, head, and record address of the record to
be altered.

Split Cylinder Disk Organizations

Split cylinder data file organizations are supported by DITTO for the Disk
Dump Unblocked (DDU), Disk Dump Reblocked (DDR), and the Disk Record Scan
(DRS) functions. Split cylinder operations are denoted by replacing the
first D in the above function codes with an S. For these split cylinder
operations, the beginning and ending head addresses will determine the lower
and upper head boundaries for the function.

Disk Identification Change

This function will print on SYSLOG the existing disk volume serial number and
allow the user to enter on SYSLOG a new volume serial number.

Initialize Tape

Initialize tape using the standard DOS format with a tape volume serial
number entered via the console. Existing tape labels or data on the output
tape is not checked before creating the new label set.

Tape To Tape

Tape-to-tape copy including tape marks. This function will copy from 1 to 99
tape files per volume as requested via the console. One file will be copied
per control card if console operation is not being used. The first file
copied may or may not have a leading tape mark. A leading tape mark does not
itself constitute a separate file. Block count statistics will be printed on
SYSLST for control card operation, and on SYSLOG for console operation, for
each tape file copied.
Tape Record Load

This function is used to copy and selectively alter existing tape records. The user need not know the exact location (block #) of the input tape record to be altered.

The first portion of the Tape Record Load function locates the desired record. The user enters an estimated block location and the parameter "F" to denote copying of tape in a forward position. When the desired record is found, it is printed on SYSLST and an option given to alter the record. All input records prior to the record to be altered have been copied onto the output tape. If the printed record is not the desired record, the user can again enter the # of blocks and direction to reposition the input and output tapes. If the user has gone past the desired record, a parameter of "B" should be entered with a block count. This will cause both input and output tapes to be backspaced the required number of blocks. This count includes the block just printed. When the desired record is located, the user may alter the length and/or data content in a manner similar to the Disk Record Load function. After all alterations are completed, replies of "9999" and "F" (9999 blocks to be copies in a forward direction) will usually be sufficient to finish copying the tape file.

Tape Record Scan

This function allows the user to specify a logical record size and a 1 to 35 position scan argument for an input tape. DITTO will read and internally deblock input records until the desired record is located. At that time the entire physical record with its associated block number will be printed and the operation terminated. The tape will not be rewound.

Tape Print

Four printing formats are available for input tapes. Tape data may be printed in character format only, or in character and vertical hex format. Physical tape records may be printed as they exist (unblocked) or may be deblocked into logical records. Block and record counts are printed with the associated records.

For control card operation, an optional NBLKS=nnnn parameter may be specified. "nnnn" specifies the maximum number of physical tape record blocks to be printed. If "nnnn" is greater than the existing number of blocks in the tape file, only the existing tape file will be printed. A leading tape mark will be automatically skipped.
Print SYSLST Tapes

Tape to printer unblocked using the first position of the tape record as the carriage control character. Two carriage control formats are available. Type A forms control (TFA) uses the carriage control character as the CCW operation code. DITTO performs its printing using Type A. Type D forms control (TFD) is used by DOS for compiler listings, etc.

Tape Control

Eight tape control functions are available either by control cards or console communications. These control functions are: write tape mark; rewind tape; rewind and unload tape; forward space or backspace file; forward space or backspace records; and erase record. The erase record will remove approximately 4" of tape data.

Card To Tape

Card image to tape with or without reblocking. Job control cards, including /* and /& cards, may be interspersed in the input data (see Card Input Assignment). A tape mark will be written after the last card block on the tape file.

Tape To Card

Tape to card for blocked or unblocked 80 character logical records. A leading tape mark will be bypassed if present. If the input tape record is not a multiple of 80 bytes, the function will terminate.

Card To Printer

Cards may be listed on SYSLST in either character or character and vertical hex format. Job Control cards including /* and /& statements may be interspersed in the input data (see Card Input Assignment).

Card To Card

Two card to punch options are available. CCU is an 80/80 reproduction of the input cards. CCS is a card-to-card copy with sequence numbers and deck identification name added. Three decktypes are available.
DECKTYPE=COB is used to sequence and name COBOL source decks. Sequence numbers will be placed in CC 1-6 and a 0-8 position deckname placed in CC 73-80.

DECKTYPE=RPG is used for RPG source decks. Sequence numbers will be placed in CC 1-5 and a 0-6 position deckname placed in CC 75-80.

DECKTYPE=BAL is used for Assembler and FORTRAN Source decks, Job Control decks, etc. Sequence numbers will be placed in CC 77-80 and a 0-4 position deckname placed in CC 73-76.

Decknames will be left-justified and padded with blanks if less than the required length. /* and /& cards may be interspersed in the input data (see Card Input Assignment).

End Of Job

All functions, whether control card or console initiated, must issue the function code "EOJ" to return control to Job control. For control card operations, this "EOJ" card must be the last function requested.

Tape Errors

All tape errors will initially be handled by the DOS supervisor. For input tape errors that are uncorrectable, DITTO will transfer control to a console routine. This routine will display the record as read, and allow the operator to accept, bypass, or alter the data and/or length of the tape record before returning control to the original operation. This option is not available for remote programmer control card operation.

Card Input Assignment

If the user does not desire to include /* and/or /& cards as a part of his input data, normal DOS procedures should be followed. For control card operations, the end of the input deck is signaled with a /* card. This will denote end-of-file for the particular operation and is consistent with DOS operating procedures. The /* card may be followed with additional DITTO operations in a similar manner. Operations which do not require data card input should not contain a /* card. The last DITTO operation must be an EOJ card and should be followed by a /* card if other DOS job steps follow, or by a /& card if a new job is to be initiated by Job Control. Console operations for data card input may be performed in a similar manner. For console operations which do not have a stacked job input stream, /* and /& cards may be omitted. End-of-file on card input will be denoted by physical end-of-file on the input device.
The DOS Supervisor monitors all cards read by the problem program from SYSIPT or SYSRDR. If the problem program attempts to read past a /* statement using either of these system logical units, the Supervisor cancels the job. The Supervisor does not, however, monitor data from a programmer logical unit. To read /* statements, DITTO must, therefore, read card input data from a programmer logical unit.

The inclusion of /* and /* cards in the input is available under console operation only. To initiate a card input operation with /* and/or /* cards in the input data, the operator must substitute an "&" for the first "C" in the function code (i.e. CCS becomes &CS). A blank card in the input stream denotes end-of-file.

CONTROL CARD OPERATION

If the user wishes to remove the decision requirements from the console operator, he may prepare control cards and submit the task in a normal Job Stream environment.

To denote control card operation the user must submit between the Job card and Exec card a // UPSI 1 card. DITTO will test the communications region to determine whether control card or console operation is desired. Depressing the interrupt key at any time overrides the control card option.

The required control card information can be contained on one card. Each control card is treated as a new operation. The control card format is as follows:

```
CC 1 -7 $DITTO
CC 10 -12 Function Code (see table)
CC 16 Parameter 1, ..., Parameter N
```

Parameters are in standard key word format. Each parameter must be separated with a comma with no embedded blanks. A blank stops the card scan. Lower case letters denote user implied information. Parenthesis () denote optional parameters. Refer to the "Parameter Requirements" for a listing of parameters associated with each function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT=SYSnnn</td>
<td>Logical input device</td>
</tr>
<tr>
<td>OUTPUT=SYSnnn</td>
<td>Logical output device</td>
</tr>
<tr>
<td>BEGIN=ccchh</td>
<td>Lower disk extent</td>
</tr>
<tr>
<td>END=ccchh</td>
<td>Upper disk extent</td>
</tr>
<tr>
<td>NBLKS=nnnn</td>
<td>Number of tape blocks</td>
</tr>
<tr>
<td>RECSIZE=nnnn</td>
<td>Logical record size</td>
</tr>
<tr>
<td>BLKFACTOR=nnnn</td>
<td>Output blocking factor (CTR)</td>
</tr>
<tr>
<td>DECKTYPE=xxx</td>
<td>CCS decktype</td>
</tr>
<tr>
<td>DECKNAME=x....x</td>
<td>CCS deckname (0-8) characters. Decknames which are less in length than required will be left</td>
</tr>
</tbody>
</table>

CSD.SFW,DITTO
justified and padded with blanks.

**PARAMETER REQUIREMENTS**

<table>
<thead>
<tr>
<th>CC 1-7</th>
<th>CC 10-12</th>
<th>C 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$DITTO$$</td>
<td>DDU</td>
<td>INPUT=SYSnnn, BEGIN=ccchh, END=ccchh</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>DDR</td>
<td>INPUT=SYSnnn, BEGIN=ccchh, END=ccchh, RECSIZE=nnnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>SDU</td>
<td>INPUT=SYSnnn, BEGIN=ccchh, END=ccchh</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>SDR</td>
<td>INPUT=SYSnnn, BEGIN=ccchh, END=ccchh, RECSIZE=nnnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>TPU</td>
<td>INPUT=SYSnnn(, NBLKS=nnnn )</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>TPR</td>
<td>INPUT=SYSnnn, RECSIZE=nnnnn(, NBLKS=nnnn )</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>THU</td>
<td>INPUT=SYSnnn(, NBLKS=nnnn )</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>THR</td>
<td>INPUT=SYSnnn, RECSIZE=nnnnn(, NBLKS=nnnn )</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>TFA</td>
<td>INPUT=SYSnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>TFD</td>
<td>INPUT=SYSnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>TCR</td>
<td>INPUT=SYSnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>CTU</td>
<td>OUTPUT=SYSnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>CTR</td>
<td>OUTPUT=SYSnnn, BLKFACTOR=nnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>TTU</td>
<td>INPUT=SYSnnn, OUTPUT=SYSnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>WTM</td>
<td>OUTPUT=SYSnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>REW</td>
<td>OUTPUT=SYSnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>RUN</td>
<td>OUTPUT=SYSnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>FSF</td>
<td>OUTPUT=SYSnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>BSF</td>
<td>OUTPUT=SYSnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>FSR</td>
<td>OUTPUT=SYSnnn, NBLKS=nnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>BSR</td>
<td>OUTPUT=SYSnnn, NBLKS=nnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>CPU</td>
<td>OUTPUT=SYSnnn</td>
</tr>
<tr>
<td>$$DITTO$$</td>
<td>CHU</td>
<td>OUTPUT=SYSnnn</td>
</tr>
</tbody>
</table>
CONSOLE OPERATIONS

All operator communication may be in upper or lower case. The cancel key is also implemented allowing retyping of replies in error. The end of message may be denoted by the "EOB" key or by repeatedly depressing the space bar.

Depressing the interrupt key for background operation will cause the existing function to be terminated and a new operation request displayed on the console. (See Multiprogramming Considerations for interrupting foreground operation.)

Tape address replies are entered in the form of "CUUMM" where C=channel, UU=unit address, and MM=density and mode setting. For nine-track single density tapes, blanks or zeros may be substituted for C0, or the MM parameter may be omitted.

If the operator desires the inclusion of /* and/or */ cards in the input data for card input functions (i.e. loading Job Streams on tape, etc.), he must replace the first "C" of the function code with a "/&". For these operations, a blank card denotes end-of-file. If the operator desires to cancel a program with card data input, he should interrupt DITTO and enter "CCL" on the console. This will cause the input stream to be flushed up to and including the end-of-file card.

DITTO may also be executed in the background entirely from the console by entering:

```
// JOB Anyname
// EXEC DITTO
```

After completing the desired operations, including the function "EOJ", a "READY FOR COMMUNICATIONS" message will be printed by Job Control. Reply:

```
/&
```

MULTIPROGRAMMING CONSIDERATIONS

Interrupting Foreground Operations

To interrupt DITTO when executing in a background partition, the external interrupt key on the CPU is depressed. This will cause the existing DITTO
function to be terminated and a console message issued requesting a new function.

The interrupt DITTO when executing in a foreground partition, the following procedures should be used:

1. Depress Console Request Key.
2. Message "AR READY FOR COMMUNICATIONS" will be displayed.
3. Enter "MSG Fn", where n=1 or 2 and correspond to the partition being interrupted.
4. Enter EOB (Alt. coding 5).

The existing foreground DITTO function will be terminated and a console message issued requesting a new function.

Batch Job Foreground Operation (DOS Version III Only)

Requirements for Batch Job Foreground operation are identical to those of background operations. Control card operations may be interchanged in any batch partition. System units (SYSLST, SYSIPT, SYSPCH) must be assigned if used in any of the DITTO functions.

Foreground Initiator Operation

Foreground Initiator Operation of DITTO is available in the console mode of communication only. Control card operation, due to the lack of an active communications region for the partition, is not available. Due to the lack of system units (SYSIPT, SYSPCH, SYSLST) in the foreground for DOS Version II, SYS001, SYS002 and SYS003 should be assigned for the above devices respectively. These system unit restrictions do not apply to DOS Version III users.

System Unit Assignments

DITTO will determine at load time the device types assigned to SYSLST and SYSPCH. If SYSLST is assigned to tape, 133 character records will be written with the first character being the Type A carriage control character. If SYSPCH is assigned to tape, 80 character records will be written. At EOJ, a tape mark will be written and the tape backspaced for the appropriate System Unit if data has been written.

If SYSPCH is assigned to a unit record device, the appropriate write command for either a 2500 series, 1442 N1, or 1442 N2 punch will be used based on the user's supervisor. No DITTO modifications should be made for these devices.
<table>
<thead>
<tr>
<th>Code</th>
<th>Alternate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCU</td>
<td>CC</td>
<td>Card to Card</td>
</tr>
<tr>
<td>CCS</td>
<td>--</td>
<td>Card to Card with sequence numbers and deck name identification</td>
</tr>
<tr>
<td>CPU</td>
<td>CP</td>
<td>Card to Printer in character format</td>
</tr>
<tr>
<td>CHU</td>
<td>CH</td>
<td>Card to Printer in character and vertical hexadecimal format</td>
</tr>
<tr>
<td>CTU</td>
<td>CT</td>
<td>Card to Tape Unblocked</td>
</tr>
<tr>
<td>CTR</td>
<td>--</td>
<td>Card to Tape Reblocked</td>
</tr>
<tr>
<td>TCR</td>
<td>TC</td>
<td>Tape to Card unblocked or blocked</td>
</tr>
<tr>
<td>TPU</td>
<td>TP</td>
<td>Tape to Printer unblocked in character format</td>
</tr>
<tr>
<td>TPR</td>
<td>--</td>
<td>Tape to Printer reblocked in character format</td>
</tr>
<tr>
<td>THU</td>
<td>TH</td>
<td>Tape to Printer unblocked in character and vertical hexadecimal format</td>
</tr>
<tr>
<td>THR</td>
<td>--</td>
<td>Tape to Printer reblocked in character and vertical hexadecimal format</td>
</tr>
<tr>
<td>TTU</td>
<td>TT</td>
<td>Tape to Tape (1 to 99 tape files)</td>
</tr>
<tr>
<td>*INT</td>
<td>--</td>
<td>Initialize Tape - DOS standard format</td>
</tr>
<tr>
<td>*TRL</td>
<td>--</td>
<td>Tape Record Load. Copy tape to tape with altering of length and/or data of specific records</td>
</tr>
<tr>
<td>*TRS</td>
<td>--</td>
<td>Tape Record Scan. Search tape for logical record using 1-35 position scan argument</td>
</tr>
<tr>
<td>TFA</td>
<td>--</td>
<td>Tape to Printer using Type A forms control</td>
</tr>
<tr>
<td>TFD</td>
<td>--</td>
<td>Tape to Printer using Type D forms control</td>
</tr>
<tr>
<td>WTM</td>
<td>--</td>
<td>Write Tape Mark</td>
</tr>
<tr>
<td>REW</td>
<td>--</td>
<td>Rewind Tape</td>
</tr>
<tr>
<td>RUN</td>
<td>--</td>
<td>Rewind and Unload Tape</td>
</tr>
<tr>
<td>FSF</td>
<td>--</td>
<td>Forward Space tape file</td>
</tr>
</tbody>
</table>

CSD.SFW.DITTO 13

01-01-06
<table>
<thead>
<tr>
<th>CODE</th>
<th>ALTERNATE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSF</td>
<td>--</td>
<td>Backspace tape file</td>
</tr>
<tr>
<td>FSR</td>
<td>--</td>
<td>Forward Space tape record</td>
</tr>
<tr>
<td>LSR</td>
<td>--</td>
<td>Back space tape record</td>
</tr>
<tr>
<td>ERG</td>
<td>--</td>
<td>Erase Record Gap</td>
</tr>
<tr>
<td>DID</td>
<td>--</td>
<td>Disk Identification change (change volume serial number)</td>
</tr>
<tr>
<td>DRL</td>
<td>--</td>
<td>Disk Record Load. Alter key and/or data portion of any disk record.</td>
</tr>
<tr>
<td>DDU</td>
<td>DD</td>
<td>Disk Dump. Disk to printer in character and vertical hexadecimal format unblocked.</td>
</tr>
<tr>
<td>DDR</td>
<td>--</td>
<td>Disk Dump with reblocking of data records into logical record length.</td>
</tr>
<tr>
<td>DRS</td>
<td>--</td>
<td>Disk Record Scan. Scan disk for matching key or data argument or for EOF.</td>
</tr>
<tr>
<td>SDU</td>
<td>SD</td>
<td>Disk Dump for split cylinder files</td>
</tr>
<tr>
<td>SDR</td>
<td>--</td>
<td>Disk Dump reblocked for split cylinder files</td>
</tr>
<tr>
<td>SRS</td>
<td>--</td>
<td>Disk Record Scan for split cylinder files</td>
</tr>
<tr>
<td>EOF</td>
<td>--</td>
<td>Write disk end-of-file record</td>
</tr>
<tr>
<td>CCL</td>
<td>--</td>
<td>Cancel card input function</td>
</tr>
<tr>
<td>EOJ</td>
<td>--</td>
<td>End of Job</td>
</tr>
</tbody>
</table>

* Denotes functions not available with control card operation.

**OPERATING SUGGESTIONS**

**DEBLOCKING TAPES**

DITTO does not provide a tape to tape with deblocking function. Blocked card image tapes (i.e., IBM supplied DTR tapes, DOS maintenance tapes, etc.), may be deblocked into an unblocked format, however, by using the tape to card function and assigning SYSPC! to the output tape. The deblocked tape may then be assigned to SYSIPT, eliminating the need in many cases of physically punching the data into cards.

CSD.SFW.DITTO

14

01-01-06
PRINTING TAPE FILES

Programmers who require listings of tape files will find the Tape Hex formats more useful than the character print functions. The Tape Hex functions provide both character and vertical hexadecimal format with a position scale for each tape record. Programmers using these functions under the remote control card operation should also make use of the NBLKS parameter. This optional parameter gives the programmer the capability of obtaining "partial" listings of tape files. This eliminates "lengthy" and "time consuming" tape printing.

PRINTING SYSLST TAPES

When SYSLST is assigned to tape, DOS Job Control statements are written as 121 character records, with the first character being the ASA FORTRAN forms-control character (Type D). DITTO and many other Type III programs which support SYSLST on tape, write 133 character records, with the first character being the CCW operation code for forms-control (Type A). Printing SYSLST tapes presents, therefore, some incompatibilities in forms-control characters to be used.

DITTO provides two (2) functions for printing SYSLST tapes using forms-control characters. TFD assumes all tape records contain the Type D forms-control character. These tape records may contain from 0 to 132 positions of data to be printed.
10. WHAT IS AN INDEXED SEQUENTIAL FILE? Before answering that question, it may be necessary to define a couple of terms. They are physical organization, and logical organization. Physical organization refers to the specific physical layout and placement of records within a file from the standpoint of the hardware. Logical organization, on the other hand, has to do with how the programmer or the operating system for that matter, "sees" the records within a file. Physical and logical organization may be exactly the same for any one file, but that is usually not the case when dealing with files which are resident on tape or on a direct access device.

An indexed sequential (ISAM) file is one in which the records are in sequence logically but not necessarily physically. It provides sequential processing capability just as any other sequential file, and in addition, provides the ability to directly access individual records. It must be resident on a direct access storage device (e.g., disk). It differs from a sequential file in that each record within the file contains a specific control field called a key. This key field must be identified to the operating system, and the key within each record must be unique; i.e., no two records in the file can have the same key, just as no two people should have the same social security number.

The total file is composed of at least three separate areas - an index area, a prime data area, and an overflow area. Though these areas are "controlled" by the operating system, the programmer can vary the size of each, and, under certain conditions, can look at what is contained in each. Normally, he is not concerned with the details of these areas.

To create or load an indexed sequential file, the records must be-in ascending sequence by key. As the records are loaded, the system builds a series of indexes, and places all of the records in the prime data area. Nothing goes into the overflow area during the create process. That area is used only when records are subsequently added to the file, and the system needs space for expansion.

The major advantage to using an indexed sequential file is the great amount of flexibility it provides. When records are added, the whole file does not need to be rewritten as is the case with a straight sequential file. Additionally, as mentioned before, direct accessing and processing of records is possible because of the indexes. Any number of records, in any order, may be read or written without processing the entire file.

Now let us look at the mechanics of creating, updating, and listing indexed sequential files using ANS COBOL. All references are to the sample program attached as Appendix A.
11. REVIEW OF ENVIRONMENT DIVISION AND DATA DIVISION REQUIREMENTS. To process ISAM files, we must describe them properly in the ENVIRONMENT DIVISION and in the DATA DIVISION.

a. ENVIRONMENT DIVISION. SELECT STATEMENT.  
(Ref: Lines 54 thru 61.)

(1) The ASSIGN clause.  
(Ref: Lines 55 and 58.)

DOS: This clause must be of the form SYSnnn-DA-unit-I in which nnn may be any system number allowed on your system, and unit may be either 2311 or 2314 or 2321. The I identifies the file as being an indexed-sequential file.

OS: This clause must be of the form DA-I-ddname where ddname is the name of the JCL DD card pointing to the indexed-sequential file. This assign identifies the file to the COBOL compiler as being an indexed-sequential file on a direct access device.

(2) The ACCESS clause. When you have an ISAM file, you have the choice of using it as a random file (Direct Access) or as a sequential file. If you wish to use the file as a random file and be able to directly access specific records, you must include the clause ACCESS IS RANDOM in the SELECT statement. If you plan to use the file as a sequential file, you may code either ACCESS IS SEQUENTIAL or omit the ACCESS clause entirely, as SEQUENTIAL is the default.  
(Ref: Line 59)

(3) The KEY clauses. The RECORD KEY IS clause identifies which field of the ISAM record is the key field. The record key must be specified for any use of an ISAM file. The NOMINAL KEY IS clause is needed whenever ACCESS IS RANDOM is specified. The nominal key is used to identify which record is to be read or written, whereas the record key identifies the one field within the record which is to be used as the key. In other words, when you want a specific record, you must ask for it by its key. This is done by moving the key of the record you want to the nominal key field, and then issuing a READ statement. When accessing an ISAM file sequentially, the nominal key is not required.  
(Ref: Lines 56, 60 and 61)

b. DATA DIVISION. FD ENTRIES. The LABEL RECORDS clause in the FD of an ISAM file must be LABEL RECORDS ARE STANDARD. ISAM files may contain only fixed-length records, which may be blocked. As with any other type of file, if the records are blocked, the BLOCK CONTAINS clause must be included as part of the FD.  
(Ref: Lines 89 thru 93 and 99 thru 103.)
12. CREATING AN ISAM FILE. (Ref: Lines 208 thru 228) We need two files to create an ISAM file - an input file within which the records, as mentioned before, must be in ascending sequence based on the record key; and an output file which is the ISAM file itself. The Environment Division entry for this file must indicate that access is sequential. The Data Division entry must show LABEL RECORDS ARE STANDARD. In the Procedure Division, the following process is required as a minimum:

a. The ISAM file must be opened as an output file.

b. The records read from the input file must be moved to the record area for the ISAM file. If reformating is desired, it should be done during this step.

c. To get each record onto the ISAM file, a WRITE statement is needed. The WRITE statement used here is slightly different from that used for straight sequential files. In the event that the records are not actually in sequence, a simple write statement would cause the program to abnormally terminate (ABEND). To prevent this, we can include an INVALID KEY clause with the WRITE statement. The format is as follows:

```
WRITE record-name FROM data-name

INVALID KEY Imperative Statement(s)
```

(Ref: Lines 219 thru 226.)

The INVALID KEY is a condition for the WRITE just as AT END is a condition for the READ, and they operate in much the same way. Until the INVALID KEY condition occurs, the statement(s) following the words INVALID KEY are ignored, and control is passed to the next sentence. If the INVALID KEY condition does occur, ONLY then will the statement(s) be executed. The reason for the INVALID KEY option is to prevent us from putting together an invalid file. The INVALID KEY condition will occur whenever we violate one of the three rules concerning building ISAM files.

Rule 1. The records which will make up the ISAM file must be in ascending sequence by their keys. If we attempt to WRITE a record that is out of sequence, an INVALID KEY condition will occur.

Rule 2. The key in each record must be unique. If we attempt to WRITE a record whose key is a duplicate of one we have already written, we also will get an INVALID KEY condition.

Rule 3. Enough disk space must be allocated to contain all records, else an invalid key condition will occur.
The statements following the words INVALID KEY are determined by what you want your program to do when an INVALID KEY condition occurs. You may want to skip over the record that caused the INVALID KEY condition; you may want to print an error message; or you may want to halt the program. Most often, the statements will be a GO TO EXIT or a PERFORM of an error routine.

d. Every record in the file should have a one character field at the very beginning reserved for a deletion code. (Ref: Lines 95 and 105) During creation of the file, the programer is responsible for moving any non-HIGH-VALUE character to this field (Traditionally LOW-VALUE is moved). Any record in the file with HIGH-VALUES in the first character will not be retrieved when processing the file sequentially.

13. UPDATING AN INDEXED SEQUENTIAL FILE. There are two ways to access an existing ISAM file. Let us look at the mechanics of each.

a. RANDOM ACCESS. Accessing a file randomly means that individual records may be selected from the file in any order. The primary benefit in using this method of processing is that not all records have to be read, and processing time can often be reduced substantially. Considerations for random access:

(1) ENVIRONMENT DIVISION. The SELECT statement for the ISAM file must include the ACCESS IS RANDOM, NOMINAL KEY, and RECORD KEY clauses. (Ref: Lines 57 thru 61.)

(2) DATA DIVISION.

(a) In describing the file (FD), the main consideration is to ensure that the block size, record size, and record format are the same as when the file was created. (Ref: Lines 99 thru 103.)

(b) The data-name associated with the NOMINAL KEY in the SELECT statement must be defined in the WORKING-STORAGE section, preferably as a Level 77 item. It must be the same size, and have the same characteristics as the item within the record which is designated as the RECORD KEY. (Ref: Lines 118, 119.)

(3) PROCEDURE DIVISION. (Ref: Lines 244 thru 287.) Any time an ISAM file is accessed randomly, there must be some means of determining which specific records of the file are to be read. Usually, another input file, called a transaction file, is used to obtain the keys of the records we need. This transaction file generally contains data to update certain records of the ISAM file, and though it may be a physically sequential file, the records
within it do not have to be in any particular order. To handle this type of situation we need to do the following in the Procedure Division:

(a) Open the transaction file as Input.

(b) Open the ISAM file for both Input and Output, because we will read records from that file, update them, and then put them back on the file. Also, we may need to add new records to the file. The OPEN statement looks like this:

    OPEN I-O ISAM-file-name

    (Ref: Line 176)

(c) A record must now be read from the transaction file. Within that record will be the key of the record to be read from the ISAM file, and it must be moved to the NOMINAL KEY field in working-storage.

(d) The ISAM record may now be retrieved by issuing a READ statement. This particular READ is slightly different from that used with sequential access. Remember, the file is being read randomly, and any record selected may be at the end of the file, in the middle, or at the beginning. In fact, any record may be read more than once in any one processing cycle. So, there is really no end to the number of records read, and an AT END clause is not required with the READ. There is a possibility though, that when we issue a READ for the file, and the system searches for the appropriate record (based on the contents of the NOMINAL KEY), that it may not find that record. If that occurs, an invalid key condition results and unless we provide for handling that condition, the program will ABEND. Fortunately, we can provide for it simply by attaching an INVALID KEY clause to the READ. The format of a READ for random access is:

    READ file-name INTO data-name

    INVALID KEY statement(s)

    (Ref: Lines 250 thru 256.)

The action of the INVALID KEY clause is the same as it was for the WRITE statement; that is, the statement(s) following the words INVALID KEY will not be executed unless an invalid key condition occurs. As stated earlier, that happens when we ask for a particular record, and that record does not exist on the file. It may be that an error of some sort has occurred possibly in source coding or keypunching - and it would then be appropriate to print an error message as well as an image (copy) of the transaction record. Or, if
the processing cycle provides for the addition of new records to the file, it may be that this transaction record is one which should be added, and we would want to perform an add-records routine. The procedures to do that will be discussed later.

(e) Assume for now that upon execution of a READ, no invalid key condition was encountered, and the record we asked for has been retrieved from the file, and is now resident in the ISAM record area of main storage. Any required changes may now be made, and when completed, the record is ready to be written back to the file. In all cases so far, when we needed to put a record onto a file, we used a WRITE statement. In this case however, we must use a REWRITE statement. The reason is that the original record is still on the file. (When we read it, a copy of the record was moved to main storage.) In order to put a "new" copy of the record onto the file, we must REWRITE rather than WRITE. The format of the REWRITE is:

```
REWRITE record-name FROM data-name
INVALID KEY statement(s)
```

(Ref: Lines 283 thru 285.)

Notice that we must attach an INVALID KEY clause to the REWRITE. The only reason for an invalid key condition to occur on a rewrite is a change in the contents of the nominal key field between the time the record was read and the time the rewrite was attempted. It is most unlikely that this will happen, but we must provide for it. If it does happen, it is best to print a message indicating what happened (INVALID KEY ON REWRITE), display the record involved, and then terminate the program, as there is evidently a logic error in the program.

(f) In addition to making changes to records, it may be that some records have to be deleted. To indicate which records should be changed, deleted, etc., we can include a type-transaction code in each transaction record. For example, a code of "C" could mean change, a "D" delete, and an "A" add. So far, we have handled only type code "C". The process of deleting records is very similar to that of changing records. The major difference is that we must move HIGH-VALUES to the deletion code field of the ISAM record, which, as pointed out in paragraph 12(d), is a one character field attached to the front of the record at file create time. No other changes need to be made in the record, and it may now be rewritten to the file just as was done with a changed record. Again, (Ref: Lines 270 thru 274) the REWRITE verb must be used instead of WRITE. Though the record is still physically on the file, from a logical standpoint it is not there. It can, however, be retrieved randomly in case it was "deleted" in error.
Now let us look at how we can add new records to an existing ISAM file. An important point we may need to reemphasize here is that the key of each record must be unique. So, if we have to add a new record to the file, it is imperative that there is not already a record on the file with the same key. We can check that in one of two ways. First, let us follow the same procedure as before. We read a record from the transaction file, move the key found there to the NOMINAL KEY area, and then issue a READ to the ISAM file. If an INVALID KEY condition occurs, the record we requested is not on the file. If the type code of the transaction is an "A" then we may add the record to the file using a WRITE statement. (A REWRITE is used only when a record which was previously read has to be put back on the file.) The format of the WRITE is the same as that used in creating the file. Another way to handle the addition of records is to read a record from the transaction file, check the type code to see if it is an "A", and if so, move the transaction record to the ISAM record area, the record key to the nominal key, and then WRITE. If that particular record is not already on the file, the WRITE will be completed successfully (in a very rare case, the file may be full, and an INVALID KEY condition will occur). If there is a record on the file with the same key, an INVALID KEY condition will occur, and it must be handled as an error.

14. SEQUENTIAL RETRIEVAL OF AN INDEXED SEQUENTIAL FILE. (Ref: Lines 305 thru 314) As mentioned before, indexed sequential files can be processed either randomly or sequentially. When updating, it is generally more advantageous to use the random access mode. (To add records to an ISAM file, random access must be used.) Sequential retrieval of records to produce, for example, a printed listing is often necessary. This paragraph will illustrate the process of retrieving sequentially first of all, an entire ISAM file, and secondly, only a portion of that same file. Again let us look at the considerations by COBOL division.

a. ENVIRONMENT DIVISION. The SELECT statement must include the RECORD KEY IS clause following the ASSIGN TO DA-I-sysname clause. The ACCESS IS SEQUENTIAL clause is optional, but should be included for documentation. If it is desired to start the retrieval process at some point other than the first record, then the NOMINAL KEY IS clause must be included also.

b. DATA DIVISION. The only factors to be considered here are that the blocking factor and record size parameters are specified correctly.

c. PROCEDURE DIVISION.

(1) The file must be opened as an input file only.

(2) If the retrieval is to start at a point other than the beginning of the file (first record), the next step must be to move the RECORD KEY of
the first record to be retrieved to the NOMINAL KEY area, and then issue a
START command. The format of the START statement is as follows:

```
START file-name

INVALID KEY imperative statement
```

This causes the ISAM file to be searched for the record whose key is con-
tained in the NOMINAL KEY area. If the record is found, a pointer is set
so that when a READ statement is given (see next paragraph), the first
record read will be the one with the given key, and each subsequent issu-
ance of a READ will result in the next record in sequence being read. If
the record requested is not found in the file then the statement following
INVALID KEY will be executed. Note that the START statement is not required
when retrieval is to begin with the first record of the file.

(3) To actually retrieve a record, a READ statement is needed.
The format of this particular READ is the same as that for a straight sequen-
tial file, i.e.,

```
READ file-name

AT END imperative statement
```

(Ref: Lines 306 thru 309)
PE-7 EXAMPLE

COBOL (Structured)

I. INTRODUCTION.
The Supply Officer at Fort Harrison has asked for you to create an ISAM File containing the current Quantity On Hand of the items stocked in the Supply Room. You are to update the ISAM file with each day's Order Records and list the ISAM File after each update.

II. DEFINITIONS.
ISAM FILE; Indexed Sequential Access Method file.

III. INPUT/OUTPUT.
INPUT: Input consists of an Order File stored on a 2314 disk pack. The file contains the Create Records for the ISAM File, followed by a blank record, followed by the days Order Records. Assign this file to SYS006. See Record Layouts 1 and 2 for a description of each type of record.

OUTPUT: Output will consist of two (2) listings.
1. A Transaction Listing showing each transaction as it is processed against the ISAM File with one of the following messages;
   A. TRANSACTION FOR DELETED RECORD-NOT PROCESSED
   B. UNMATCHED PART NUMBER
   C. DELETED RECORD
   D. NEGATIVE BALANCE ON HAND
   E. PROCESSED
   See Form Layout #1 for a description of this report.

   2. An ISAM FILE LISTING to list all records on the ISAM File by part number showing the updated Quantity On Hand. See Form Layout #2 for a description of this report.

   Assign these reports to SYS005.

IV. ASSUMPTIONS.
There will be more than one (1) page of output, so produce a heading at the top of each page.
CUSTOMER CODE will also contain a 'D' if the record is to be deleted from the ISAM File.
During the updating of the ISAM File no records are to be added.
V. METHODS.

1. Read the CREATE RECORDS and create the ISAM File. If you run out of space on the disk, or the records are out of sequence, stop the processing at that point.

2. Read the ORDER RECORDS, find the matching ISAM record, and subtract the Quantity Ordered from the Quantity On Hand. If the record has already been deleted, annotate it with MESSAGE 'A'. If a matching record is not found, annotate it with MESSAGE 'B'. If the CUSTOMER CODE is equal to a 'D', delete the record and annotate it with MESSAGE 'C'. If the updated Quantity On Hand happens to be NEGATIVE, annotate it with MESSAGE 'D'. Annotate all other transactions with MESSAGE 'E'.

3. List the ISAM File after it is updated.

Both reports will consist of twenty (20) double spaced lines per page.

VI. EDIT FACTORS.

All numeric fields will contain valid numeric data.
**RECORD LAYOUT #2**

For use of this form, see AR 18-7; the proponent agency is Office of the Assistant Vice Chief of Staff.

<table>
<thead>
<tr>
<th>SYSTEM ID</th>
<th>RECORD LENGTH</th>
<th>PREPARED BY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truck Supply System</td>
<td>80 Characters</td>
<td>SP6 D.W. Gross</td>
</tr>
</tbody>
</table>

- [ ] CARD  
- [x] DISK  
- [ ] TAPE  
- [ ] OTHER

**FILE ID**

- DISK  
- TAPE  
- OTHER

**REMARKS**

- 1 record per block

**ORDERS FILE (Create Records)**

```
<table>
<thead>
<tr>
<th>RELATIVE POSITION</th>
<th>IDENTIFICATION OF ELEMENT (FIELD)</th>
<th>ABBREVIATION</th>
<th>LENGTH/CLASS</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PART NUMBER</td>
<td>4N</td>
<td>1-4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Quantity On Hand</td>
<td>4N</td>
<td>5-8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Not Used</td>
<td>72AN</td>
<td>9-80</td>
<td></td>
</tr>
</tbody>
</table>
```

Appendix A

3
**RECORD LAYOUT**

For use of this form, see AR 18-7; the proponent agency is Office of the Assistant Vice Chief of Staff.

<table>
<thead>
<tr>
<th>RELATIVE POSITION</th>
<th>IDENTIFICATION OF ELEMENT ([FIELD])</th>
<th>ABBREVIATION</th>
<th>LENGTH/CLASS</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PART NUMBER</td>
<td></td>
<td>1N</td>
<td>1</td>
</tr>
<tr>
<td>1a</td>
<td>CLASS NUMBER (either a 1, 2, or 3)</td>
<td>3N</td>
<td>2-4</td>
<td></td>
</tr>
<tr>
<td>1b</td>
<td>Part Number</td>
<td>18AN</td>
<td>5-22</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CUSTOMER NAME</td>
<td>18AN</td>
<td>23-40</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CUSTOMER ADDRESS</td>
<td>2AN</td>
<td>41-42</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>UNIT OF ISSUE</td>
<td>3N</td>
<td>43-45</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>QUANTITY ORDERED</td>
<td>5N*</td>
<td>46-50</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>PRICE PER ITEM</td>
<td>1A**</td>
<td>51 ***</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>CUSTOMER CODE</td>
<td>29AN</td>
<td>52-80</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>NOT USED</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* DOLLARS AND CENTS FIELD
** CAN ONLY BE A BLANK OR A '?'
*** Will be a 'D' for deletion transactions

Appendix A
IDENTIFICATION DIVISION.

PROGRAM-ID. PE-7EXAMPLE.

AUTHOR. SP6DW GROSS

INSTALLATION. CSD, USAIA, FT HARRISON IN 46216.

DATE-WRITTEN. MAR 1, 1978

DATE-COMPiled. JUL 5, 1978

SECURITY. NONE

REMARKS. --------------------------

GENERAL FUNCTIONS.

This program illustrates

1 creating an ISAM file

2 randomly updating an ISAM file

3 reading and listing the updated ISAM

file sequentially.

SPECIFIC FUNCTIONS.

This program reads the create records

and builds an ISAM file. The order records

for each day are then matched against the

balance file (ISAM file), and the quantity

ordered is subtracted from the balance.

Files quantity on hand. If a negative re-

sult is reached, a message is printed to

that effect on the transaction report.

Finally this program lists the updated

balance file.

************ --------------------------
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-360-H40.
OBJECT-COMPUTER. IBM-360-H40.
SPECIAL-NAMES.
COI IS TOP-OF-PAGE.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT IN-ORDER-FILE
ASSIGN TO UT-S-SYS006.
SELECT OR-PRINTED-REPORTS
ASSIGN TO UT-S-SYS005.

***************************************************************
** NOTICE THE ASSIGN CLAUSE FOR THE FOLLOWING SELECTS. ISAM **
** FILES MUST BE ASSIGNED TO DIRECT ACCESS DEVICES, THUS THE 'DA **
** I'. ALSO YOU MUST SPECIFY WHICH FIELD IS THE RECORD'S KEY **
** FIELD. FOR RANDOM PROCESSING, THE CLAUSE ACCESS IS RANDOM, **
** AND NOMINAL KEY IS ... MUST BE SPECIFIED. **
***************************************************************

SELECT BM-BALANCE-FILE
ASSIGN TO DA-I-SYSC08
RECORD KEY IS OM01-PART-NUMBER.
SELECT BM-BALANCE-FILE-RANDOM
ASSIGN TO DA-I-SYSC08
ACCESS IS RANDOM
RECORD KEY IS OM02-PART-NUMBER
NOMINAL KEY IS WS-PART-NUMBER
Appendix
WORKING-STORAGE SECTION.

77 WS-BEGIN-POINT    PIC X(27) VALUE 'WORKING STORAGE BEGINS HERE'.

77 WS-EOF-SWITCH    PIC XXX VALUE 'OFF'.

77 WS-ERROR-SW    PIC XXX VALUE 'OFF'.


77 WS-PART-NUMBER    VALUE 0.

01 WS-HEADING-LINE-1.
  05 FILLER       VALUE SPACES.
  05 WS-HL1-TITLE VALUE SPACES.

01 WS-HEADING-LINE-2.
  05 FILLER       VALUE SPACES.
  05 FILLER       VALUE 'PART NO'.
  05 FILLER       VALUE SPACES.
  05 FILLER       VALUE 'QUANTITY'.
  05 FILLER       VALUE SPACES.
  05 WS-HL2-MESSAGE VALUE 'MESSAGE'.

01 WS-DETAIL-LINE.
  05 FILLER       VALUE SPACES.
  05 WS-DL-PART-NUMBER    VALUE SPACES.
  05 FILLER       VALUE SPACES.
  05 WS-DL-QUANTITY    VALUE SPACES.
  05 FILLER       VALUE SPACES.
  05 WS-DL-MESSAGE    PIC X(106).

01 WS-END-POINT    VALUE 'WORKING STORAGE ENDS HERE'.

Appendix A
PROCEDURE DIVISION.
  0000-DRIVER.

FUNCTIONAL REQUIREMENTS

THE DRIVER PORTION OF THIS PROGRAM DIRECTS THE EXECUTION THROUGH THE THREE PARTS OF THIS PROGRAM.

OPEN INPUT IM-ORDER-FILE
OUTPUT BM-BALANCE-FILE
PERFORM 0100-CREATE-ROUTINE THRU 0100-EXIT UNTIL WS-EOF-SWITCH EQUAL 'ON'. IF WS-ERROR-SW EQUAL 'ON' GO TO 0000-EXIT.
CLOSE BM-BALANCE-FILE.

THE FOLLOWING OPEN STATEMENT ILLUSTRATES THE 'I-O' OPTION. THIS MEANS THAT THE FILE WILL BE OPENED AS BOTH INPUT AND OUTPUT.

OPEN I-O BM-BALANCE-FILE-RANDOM
OUTPUT OR-PRINTED-REPORTS
MOVE 'OFF' TO WS-EOF-SWITCH.
MOVE 'TRANSACTION REPORT' TO WS-HL1-TITLE.
PERFORM 0200-UPDATE-ROUTINE THRU 0200-EXIT UNTIL WS-EOF-SWITCH EQUAL 'ON'.
CLOSE BM-BALANCE-FILE-RANDOM.
OPEN INPUT BM-BALANCE-FILE
MOVE 'OFF' TO WS-EOF-SWITCH.
MOVE 26 TO WS-LINE-COUNTER.
MOVE SPACES TO WS-DL-MESSAGE.
MOVE SPACES TO WS-HL2-MESSAGE.
MOVE 'ISAM FILE LISTING' TO WS-HL1-TITLE.
PERFORM 0500-LIST-ROUTINE THRU 0500-EXIT UNTIL WS-EOF-SWITCH EQUAL 'ON'.
CLOSE BM-BALANCE-FILE.

0000-EXIT.
STOP RUN.
THE CREATE ROUTINE READS EACH INPUT CREATE RECORD, MOVES THE FIELDS TO THE ISAM BALANCE RECORD, AND WRITES EACH RECORD. NOTICE THE WRITE STATEMENT. WHEN WRITING TO AN ISAM FILE THE INVALID KEY CLAUSE IS USED. IN THIS CASE IF THERE IS AN INVALID KEY IT MEANS THERE IS NO MORE SPACE LEFT ON THE DISK.

0100-CREATE-ROUTINE.
READ IM-ORDER-FILE
AT END
MOVE 'ON' TO WS-EOF-SWITCH
GO TO 0100-EXIT.
IF IM01-CREATE-REC EQUAL SPACES
THEN MOVE 'ON' TO WS-EOF-SWITCH
GO TO 0100-EXIT.
MOVE LOW-VALUES TO BM01-DELETE-KEY.
MOVE IM01-PART-NUMBER TO BM01-PART-NUMBER.
MOVE IM01-QUANTITY TO BM01-QUANTITY-ON-HAND.
WRITE BM01-BALANCE-REC
INVALID KEY
DISPLAY 'FULL DISK OR TRANS OUT OF SEQ',
IM01-PART-NUMBER
CLOSE IM-ORDER-FILE
BM-BALANCE-FILE
MOVE 'ON' TO WS-EOF-SWITCH
MOVE 'ON' TO WS-ERROR-SW.
0100-EXIT.
EXIT.

THE FUNCTION OF THE UPDATE ROUTINE IS TO READ EACH ORDER RECORD, READ THE ISAM FILE RANDOMLY FINDING THE MATCHING RECORD AND ADJUST THE QUANTITY ON HAND, THEN INVALID KEY CLAUSE IS USED WITH THE READ STATEMENT, WHEN THE MATCHING RECORD IS NOT THERE, THIS CONDITION IS INVOKED.

AFTER THE QUANTITY ON HAND IS ADJUSTED, THE ISAM RECORD IS REPLACED IN ITS PLACE ON THE FILE. THE REWRITE STATEMENT IS USED FOR THIS.

0200-UPDATE-ROUTINE.
READ IM-ORDER-FILE
AT END
MOVE 'ON' TO WS-EOF-SWITCH
GO TO 0200-EXIT.
MOVE IM02-PART-NUMBER TO WS-PART-NUMBER.
READ BM-BALANCE-FILE-RANDOM
INVALID KEY
MOVE IM02-PART-NUMBER TO WS-DL-PART-NUMBER
MOVE IM02-QUANTITY-ORDERED TO WS-DL-QUANTITY
MOVE 'UNHATCHED PART NUMBER' TO WS-DL-MESSAGE
PERFORM 0300-PRINT-Routine THRU 0300-EXIT
GO TO 0200-EXIT.

IF BM02-DELETE-KEY EQUAL HIGH-VALUES
THEN MOVE 'TRANSACTION FOR DELETED RECORD- NOT PROCESSED'
TO WS-DL-MESSAGE
MOVE IM02-PART-NUMBER TO WS-DL-PART-NUMBER
MOVE IM02-QUANTITY-ORDERED TO WS-DL-QUANTITY
PERFORM 0300-PRINT-Routine THRU 0300-EXIT
GO TO 0200-EXIT.

IF IM02-CUSTOMER-CODE EQUAL '0'
THEN MOVE HIGH-VALUES TO BM02-DELETE-KEY
MOVE 'DELETED RECORD' TO WS-DL-MESSAGE
MOVE IM02-PART-NUMBER TO WS-DL-PART-NUMBER
MOVE IM02-QUANTITY-ORDERED TO WS-DL-QUANTITY
PERFORM 0300-PRINT-Routine THRU 0300-EXIT
REWRITE BM02-BALANCE-REC-R
INVALID KEY
DISPLAY 'RECORD NOT DELETED',
IM02-PART-NUMBER
GO TO 0200-EXIT.

SUBTRACT IM02-QUANTITY-ORDERED FROM BM02-QUANTITY-ON-HAND.
MOVE BM02-PART-NUMBER TO WS-DL-PART-NUMBER.
MOVE BM02-QUANTITY-ON-HAND TO WS-DL-QUANTITY.
IF BM02-QUANTITY-ON-HAND IS LESS THAN 0
THEN MOVE 'NEGATIVE BALANCE ON HAND' TO WS-DL-MESSAGE
ELSE
MOVE 'PROCESSED' TO WS-DL-MESSAGE.
PERFORM 0300-PRINT-Routine THRU 0300-EXIT.
REWRITE BM02-BALANCE-REC-R
INVALID KEY
DISPLAY BM02-PART-NUMBER, WS-PART-NUMBER.
0200-EXIT.
EXIT.
0300-PRINT-Routine.
IF WS-LINE-COUNTER GREATER THAN 19
THEN PERFORM 0400-HEADER-Routine THRU 0400-EXIT.
WRITE OR01-PRINT-LINE FROM WS-DETAIL-LINE
AFTER ADVANCING 2 LINES.
ADD 1 TO WS-LINE-COUNTER.
MOVE SPACES TO WS-DETAIL-LINE.
0300-EXIT.
0300-EXIT.
0400-HEADER-Routine.
MOVE ZERO TO WS-LINE-COUNTER.
WRITE OR01-PRINT-LINE FROM WS-HEADING-LINE-1
AFTER ADVANCING TOP-OF-PAGE.
WRITE OR01-PRINT-LINE FROM WS-HEADING-LINE-2
AFTER ADVANCING 2 LINES.
0400-EXIT.
0400-EXIT.
0500-LIST-Routine.
READ BM-BALANCE-FILE
AT END
MOVE 'ON' TO WS-EOF-SWITCH
GO TO 0500-EXIT.
MOVE BM01-QUANTITY-ON-HAND TO WS-DL-QUANTITY.
MOVE BM01-PART-NUMBER TO WS-DL-PART-NUMBER.
PERFORM 0300-PRINT-ROUTINE THRU 0300-EXIT.
EXIT.
<table>
<thead>
<tr>
<th>PART NO</th>
<th>QUANTITY</th>
<th>MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1941</td>
<td>-35</td>
<td>NEGATIVE BALANCE ON HAND</td>
</tr>
<tr>
<td>1268</td>
<td>993</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1767</td>
<td>2513</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1513</td>
<td>6749</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1052</td>
<td>5048</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1223</td>
<td>-9</td>
<td>NEGATIVE BALANCE ON HAND</td>
</tr>
<tr>
<td>1454</td>
<td>4256</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1689</td>
<td>9903</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1169</td>
<td>394</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1491</td>
<td>154</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1767</td>
<td>2507</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1268</td>
<td>981</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1513</td>
<td>6740</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1941</td>
<td>-285</td>
<td>NEGATIVE BALANCE ON HAND</td>
</tr>
<tr>
<td>1655</td>
<td>117</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1689</td>
<td>9901</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1655</td>
<td>53</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1454</td>
<td>4239</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1052</td>
<td>5047</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1513</td>
<td>6738</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>PART NO</td>
<td>QUANTITY</td>
<td>MESSAGE</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>1689</td>
<td>9900</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1491</td>
<td>129</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1767</td>
<td>2499</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1454</td>
<td>4237</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1169</td>
<td>384</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>1223</td>
<td>-29</td>
<td>NEGATIVE BALANCE ON HAND</td>
</tr>
<tr>
<td>2086</td>
<td>9742</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>2019</td>
<td>9299</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>2652</td>
<td>426</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>2152</td>
<td>7341</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>2925</td>
<td>1415</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>2068</td>
<td>927</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>2529</td>
<td>5827</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>2019</td>
<td>9298</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>2652</td>
<td>414</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>2152</td>
<td>7332</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3109</td>
<td>999</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3327</td>
<td>1501</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3007</td>
<td>3579</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3318</td>
<td>7800</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>PART NO</td>
<td>QUANTITY</td>
<td>MESSAGE</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>3765</td>
<td>4987</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3082</td>
<td>9921</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3007</td>
<td>3</td>
<td>DELETED RECORD</td>
</tr>
<tr>
<td>3007</td>
<td>3576</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3007</td>
<td>2</td>
<td>TRANSACTION FOR DELETED RECORD—NOT PROCESSED</td>
</tr>
<tr>
<td>3318</td>
<td>7799</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3327</td>
<td>1498</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3765</td>
<td>4986</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3109</td>
<td>959</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3082</td>
<td>9920</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3327</td>
<td>1490</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>3765</td>
<td>4985</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>9999</td>
<td>3</td>
<td>UNMATCHED PART NUMBER</td>
</tr>
<tr>
<td>3109</td>
<td>938</td>
<td>PROCESSED</td>
</tr>
<tr>
<td>PART NO</td>
<td>QUANTITY</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>1052</td>
<td>5047</td>
<td></td>
</tr>
<tr>
<td>1169</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>1223</td>
<td>-29</td>
<td></td>
</tr>
<tr>
<td>1268</td>
<td>981</td>
<td></td>
</tr>
<tr>
<td>1454</td>
<td>4237</td>
<td></td>
</tr>
<tr>
<td>1491</td>
<td>129</td>
<td></td>
</tr>
<tr>
<td>1513</td>
<td>6738</td>
<td></td>
</tr>
<tr>
<td>1655</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>1689</td>
<td>9900</td>
<td></td>
</tr>
<tr>
<td>1767</td>
<td>2499</td>
<td></td>
</tr>
<tr>
<td>1941</td>
<td>-285</td>
<td></td>
</tr>
<tr>
<td>2019</td>
<td>9298</td>
<td></td>
</tr>
<tr>
<td>2068</td>
<td>927</td>
<td></td>
</tr>
<tr>
<td>2555</td>
<td>9742</td>
<td></td>
</tr>
<tr>
<td>2152</td>
<td>7332</td>
<td></td>
</tr>
<tr>
<td>2529</td>
<td>5827</td>
<td></td>
</tr>
<tr>
<td>2652</td>
<td>414</td>
<td></td>
</tr>
<tr>
<td>2925</td>
<td>1415</td>
<td></td>
</tr>
<tr>
<td>3082</td>
<td>9920</td>
<td></td>
</tr>
<tr>
<td>3109</td>
<td>938</td>
<td></td>
</tr>
<tr>
<td>PART NU</td>
<td>QUANTITY</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>3318</td>
<td>7799</td>
<td></td>
</tr>
<tr>
<td>3327</td>
<td>1490</td>
<td></td>
</tr>
<tr>
<td>3765</td>
<td>4985</td>
<td></td>
</tr>
</tbody>
</table>
Assembler Language

Practical Exercises

ALC Team, Software Division
Data Processing Department
United States Army Institute
of Administration
A. Write the statements necessary to define the heading shown on the attached form layout. (Form Layout 1)

B. Write the statements necessary to define the detail line shown on the attached form layout. (Form Layout 1)

C. Write the statements necessary to define an input area corresponding to the format shown on the attached card layout. (Card Layout 1)

D. Write the statement to allocate enough space for the printer output area.
### Multiple Card Layout

**card layout 1**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Rank</th>
<th>Date of Rank</th>
<th>Not Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
</tr>
<tr>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>50</td>
</tr>
<tr>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
</tr>
<tr>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
</tr>
<tr>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
</tr>
<tr>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
</tr>
<tr>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
</tr>
<tr>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
</tr>
<tr>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
</tr>
<tr>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>90</td>
</tr>
<tr>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
</tr>
<tr>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
</tr>
<tr>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
<td>105</td>
</tr>
</tbody>
</table>

**Instructional Use Only**

For use of this form, see AR 16-2, the cognizant agency is Office of the Assistant Vice Chief of Staff.
One of the programmers in your shop has left you with a partially written ALC program. He has completed the logic portion of the program and all that remains to be written is the declarative section and the DTF's.

You are to define the following areas in storage:

1. A one byte field named SPACE initialized to blank.
2. A 132 byte field named OUTPUT, no initial value is needed. This field will be used to hold character information.
3. An 80 byte field called INPUT. This field should be defined with a $ duplication factor.
4. A 79 byte field with no name attached. This field will hold character information, and no initial value is needed.
5. A one byte field named CODE. It will hold character data and needs no initial value.
6. A one byte field called DASH initialized to the value '-'.

The following DTF's are also needed:

1. A DTFFR. The I/O area is OUTPUT, the device address is SYSLST, the file name is PRINT, and the CNTRL macro will be used, the block size should be 132.

2. A DTFCO. The I/O area is INPUT, the device address is SYSIPT, the file name is CARD, and the end of file address is ENDJOB.

The above storage areas and DTFs should be coded in the order shown above. Once you have done the required coding, keypunch the cards and submit them for execution. The procedural part of the program will be supplied by the student job stream as will the data cards.
PERSONNEL LISTING

ALC PE #3

Assemble:  A01
Execute:  A03

You are required to write an ALC program to list all of the people in your unit.

INPUT: The input has the same format as the input for ALC PE #1. The logical name for the card reader is SYSIPT.

OUTPUT: The heading and detail lines will have the same format as those for ALC PE #1. You may assume that all output will fit on one page. The logical name for the printer is SYSLST. Double space all output.
I. Introduction: You are to write a program that will print a listing of the people assigned to the office. This listing will show each individual's name, phone number, home address, and marital status.

II. Definitions: SCODE = The record selection code  
MCODE = The marital status code

III. Input/Output:

A. Input: The input will be the personnel master file which is on cards (see attached card layout). Device address = SYSIPT.

B. Output: The output will be a printed report on the 1403 line printer (see the attached form layout). Device address = SYSLST.

IV. Assumptions:

A. The listing will fit on one page
B. The MCODE will be only an M, S, or D

V. Methods:

A. Provide a title and column headings as shown on the form layout
B. List only those personnel that have an SCODE of zero. Use the CLC instruction to check the SCODE
C. Using an MVI instruction, insert a hyphen between the third and fourth digit of the phone number on the printed report. This will require breaking the phone number down into two parts.
D. Determine the individual's marital status using the CLI instruction. If the code is an "M", print "MARRIED" in the STATUS field; if the code is an "S", print "SINGLE"; and if the code is a "D", print "DIVORCED".

VI. Edit Factors: None
<table>
<thead>
<tr>
<th>Name</th>
<th>Phone Number</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assemble: A01  
Execute:  A03

You are to write an ALC program to produce a listing of all personnel in the battalion who have completed more than 35 months of service.

INPUT: The input is a personnel master file presently stored on punched cards. See Card Layout 3.

OUTPUT: 1) The output is on a 132 print position printer.
2) Print a heading at the top of each new page.
3) Only 10 detail lines should be printed per page.
4) Print a total line containing the total personnel with 36 or more months of service.
5) Double space all lines except triple space between the total line and the last line.
6) See Form Layout 3 for various line formats.

Assumptions: Not more than 999 persons will have more than 35 months of service. Since we can record only full months of service, the person must have 36 months of service or more to qualify.

Methods: a) Read each card and print a detail line only for personnel with over 35 months of service.

b) Keep a count of the total number of personnel with more than 35 months of service. This total will be part of the final total line.

c) Do not edit the months of service figure. However, the total personnel figure should be edited with leading nonsignificant zeros suppressed except when the figure is 000 in which case the total figure should print as a single zero.
### Card Layout

<table>
<thead>
<tr>
<th>Social Security Number</th>
<th>Name</th>
<th>ETS</th>
<th>Date of Rank</th>
<th>Not Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Rank</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Battalion</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Company</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company</th>
<th>Rank</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependents</td>
<td>Pay Grade</td>
<td>Months of Service</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PAYROLL EXERCISE

I. Introduction: You are to write a program that will calculate the net pay and withholding tax for each individual within the company. The report that the program generates will list each individual and the information pertaining to his pay. It will also show the total tax withheld for each department.

II. Definitions: None

III. Input/Output:

   A. Input: The input is on cards which are sorted by department. The pay and tax rate fields have two implied decimal points (see attached card layout for input data format). Device address = SYSIPT.

   B. Output: The output will be on a 1403 line printer and will consist of column headings on each page, detail lines, and a total line for each department. Start each department on a new page (see attached form layout for output specifications). Device address = SYSLST.

IV. Assumptions:

   Each department will fit on one page. The total tax figure will not exceed $99999.99.

V. Methods:

   A. For each record calculate the withholding tax as follows:

      Withholding tax = Tax Rate x Gross Pay

   B. Calculate Net Pay as follows:

      Net Pay = Gross Pay - Withholding Tax

   C. Keep track of the total taxes for each department. This will be part of the total line.

   D. Zero suppress all numeric output fields. The Gross Pay, Withholding Tax, and Net Pay fields should have decimal points, commas, and fixed dollar signs as shown on the form layout.

VI. Edit factors: None
MULTIPLE CARD LAYOUT

Card Layout 5

<table>
<thead>
<tr>
<th>DEPT</th>
<th>NAME</th>
<th>GROSS PAY</th>
<th>TRATE</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>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
</tr>
<tr>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
</tr>
<tr>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
</tr>
<tr>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
</tr>
<tr>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
</tr>
<tr>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
</tr>
<tr>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
</tr>
<tr>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
</tr>
<tr>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
</tr>
</tbody>
</table>

TRATE = Tax Rate
I. Introduction: The Fix or Repair Daily Corporation has decided to start selling by the case instead of breaking a case to sell individual items. They want you to write a program that will produce a report showing the number of full cases and the number of loose items for each stock number.

II. Definitions: Loose Items = the remainder of the division of

Number of Items;
Number per Case

III. Input/Output:

A. Input: The input will be The Current Inventory File which is kept on cards (see card layout).

Device address = SYSIPT.

B. Output: The output will be a printed report on the 1403 line printer (see the form layout for format and spacing).

Device address = SYSLST.

IV. Assumptions: None.

V. Methods:

A. Provide main headings with a page number and column headings at the top of each page.

B. Edit all numeric fields to ensure that they contain numeric information.

1. If they do not, bypass all calculations and print out the card with fields in the appropriate positions. Insure that the Loose Items and Number of Cases fields are blank. The word "ERROR" should also be printed on the same line.

2. If they do, calculate the Number of Cases as follows:

   Number of Cases = \frac{Number of Items}{Number per Case}

3. Print a detail line consisting of Stock Number, Item Name, Number of Items, Number of Cases, and Number of Loose Items editing as shown on the form layout.

C. Print only 20 double spaced detail lines per page.

VI. Edit Factors: Edit all numeric input fields to ensure that they contain numeric information.
**Multiple Card Layout**

For use of this item, see AR 18-7, the proper agency is Office of the Assistant Vice Chief of Staff

### Card Layout 6

<table>
<thead>
<tr>
<th>STOCK NUMBER</th>
<th>ITEM NAME</th>
<th>NUMBER OF ITEMS</th>
<th>I/C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**I/C = Items per case**

**Page No:** 1

**Page Number:** 1

**Page of:** 20

**Item Per Case:**

<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>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
</tr>
<tr>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
</tr>
<tr>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
</tr>
<tr>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
</tr>
<tr>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
</tr>
<tr>
<td>STOCK #</td>
<td>ITEM NO.</td>
<td>QTY</td>
<td># OF PKGS</td>
<td># OF CANS</td>
<td># OF CANS,title</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>1000</td>
<td>1000</td>
<td>1</td>
<td>225,000</td>
<td>1</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>2000</td>
<td>2</td>
<td>450,000</td>
<td>2</td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>5000</td>
<td>3</td>
<td>675,000</td>
<td>3</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td>10000</td>
<td>4</td>
<td>900,000</td>
<td>4</td>
<td>D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FOR INSTRUCTIONAL PURPOSES ONLY
BLOOD TYPE

ALC PE 8

Assemble: A01
Execute: A05

You are required to determine the number of people with blood
types A, B, AB, and 0 for a department in the Institute.

INPUT: 1) The input consists of two files: A sequential disk master
file and a card file.

2) The format of the sequential disk records is given in
on disk Layout. The file has a blocking factor of 3.

3) The card format is shown on Card Layout 7.

4) The blood type field might look like the following for
example:

record position 31 79

ABCAABBOOCCCAAAAABOOOOOAC...ABB

The blood type codes have the following meaning:

<table>
<thead>
<tr>
<th>Blood Code</th>
<th>Blood Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>AB</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

Position 80 has a code. If it is anything other
than the letter 'B', ignore that disk record.

5) The records are sorted by departments and one department
may be on more than one record.

6) The card file contains one card with two numbers - a low
key and a high key; these two numbers should be used to
select the disk records to be processed. Only if the
following condition exists should the disk record be processed:

low key ≤ disk record position 1-10 ≤ high key

If the condition does not exist, ignore that record.
OUTPUT: 1) The output will be on a 132 print position printer and consist of six lines for each organization.

2) Each organization should be on a separate page with the format given on Form Layout 7.

Assumptions: a) When the disk file was created, the name assigned to the file through Job Control was DISKIN. The device address is SYS006.

b) Ignore all illegal blood types found.

c) The totals can be expected to be at most 5 digits long.

d) Edit the totals by suppressing leading non-significant zeros.
### Tape Layout

For use of this form, see AR 18-1; the proponent agency is Office of the Comptroller of the Army.

<table>
<thead>
<tr>
<th>No., Length, Block</th>
<th>Organization Number</th>
<th>Organization Name</th>
<th>1 to 49 Blood Type Code Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>80 x 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Blood Type Entries (cont'd)

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

### Multiple Card Layout

For use of this form, see AR 18-1; the proponent agency is Office of the Assistant Vice Chief of Staff.

<table>
<thead>
<tr>
<th>Card Layout 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Key</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

---

For DA Form 3166, US Army P&R 294-63 10-4, "FOR INSTRUCTIONAL PURPOSES ONLY" D-500
I. **Introduction:** You are required to write a program that will edit a Transaction File and a Master File to ensure that they meet certain requirements. These files, if there are no edit errors, will be used as input for another program.

II. **Definitions:** None.

III. **Input/Output:**

- **Input:** The Transaction File will be on cards (see card layout for specifications). The Master File will be on sequential disk. There are three records per block (see disk layout for specifications). Card is on SYSIPT, Disk is on SYS006, File name for Disk is MASTIN.
- **Output:** A printed report will be produced to show the status of the different records (see form layout for specifications). Device address = SYSLST.

IV. **Assumptions:** There will be one, and only one, transaction record per master record.

V. **Methods:**

1. Print a main heading with page number and column headings at the top of each page.

2. Print 15 double spaced detail lines per page.

3. Set up a one byte switch to be used when you edit the input.

4. Edit the classification field on the transaction card to find out whether it is blank, numeric, alphabetic, or alpha numeric. One of the first four bits of the switch in Step 3 above will be turned on to indicate what type of field it is.

5. Do the same thing for the classification field on the master record, except that one of the last four bits will be turned on to indicate what type of field it is.
6. The code on the transaction card will tell you what type this field should be.

<table>
<thead>
<tr>
<th>CLASSIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE OF CODE</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

7. If either or both of the fields are not the appropriate type, print the fields in the correct print positions along with the word "ERROR"

8. If both the fields are the appropriate type, print the fields in the correct print positions.

VI. **Edit Factors: None**
The following definitions of terms apply to ALC PE #4. These are further explained by the series of examples that follow the definitions.

1. BLANK - this means that all 10 positions of the field are blanks.

2. NON-BLANK - This means that at least 1 and maybe as many as all 10 positions of the field contain anything other than a blank.

3. NUMERIC - This means that all 10 positions of the field contain only the digits 0 thru 9.

4. ALPHABETIC - This means that all 10 positions of the field contains only the letters A thru Z, or blanks, however, all 10 positions may not be blanks.

**EXAMPLES:**

<table>
<thead>
<tr>
<th>DATA FIELD</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCDEFGHILJ</td>
<td>ALPHABETIC, NON BLANK</td>
</tr>
<tr>
<td>12345678</td>
<td>NON BLANK</td>
</tr>
<tr>
<td>.(AXC6.*)/</td>
<td>NON BLANK</td>
</tr>
<tr>
<td>1234567890</td>
<td>NUMERIC, NON BLANK</td>
</tr>
<tr>
<td>***************</td>
<td>BLANK</td>
</tr>
<tr>
<td>*************</td>
<td>NON BLANK</td>
</tr>
<tr>
<td>*************</td>
<td>NON BLANK</td>
</tr>
<tr>
<td>XXABCX6XX</td>
<td>NON BLANK</td>
</tr>
<tr>
<td>2468013579</td>
<td>NUMERIC, NON BLANK</td>
</tr>
<tr>
<td>JOHNDOES</td>
<td>ALPHABETIC, NON BLANK</td>
</tr>
<tr>
<td>TITLE</td>
<td>MASTER ERUI REPORT</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------</td>
</tr>
<tr>
<td>HERON</td>
<td></td>
</tr>
<tr>
<td>DETAIL</td>
<td></td>
</tr>
</tbody>
</table>

**Transactions**

<table>
<thead>
<tr>
<th>Item</th>
<th>Code</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>XX000</td>
<td>X</td>
<td>XXX000</td>
</tr>
<tr>
<td>XX000</td>
<td>X</td>
<td>XXX000</td>
</tr>
<tr>
<td>XXX00</td>
<td>X</td>
<td>XXX000</td>
</tr>
</tbody>
</table>

**Lines**

<table>
<thead>
<tr>
<th>Item</th>
<th>Code</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>XX000</td>
<td>X</td>
<td>XXX000</td>
</tr>
<tr>
<td>XX000</td>
<td>X</td>
<td>XXX000</td>
</tr>
<tr>
<td>XXX00</td>
<td>X</td>
<td>XXX000</td>
</tr>
</tbody>
</table>

**Instructions**

- For instructional purposes only.
I. Introduction: You are to write a program that will produce a one page report showing the current base pay for all enlisted personnel with less than 20 years service.

II. Definitions: None.

III. Input/Output:

   Input: None:
   
   Output: The output will be on a 1403 line printer (SYSLST), and will consist of a one page report as shown on the attached Form Layout.

IV. Assumptions: None.

V. Methods:

   1. Print the column heading on the top of the page.
   
   2. A macro called EMPAY has been catalogued on the system and it will generate the required nine pay tables, each table having eleven entries. Each entry of the tables will be a packed four byte field containing the old base pay. The label that you code on the macro will appear as the name of each table and will be followed by a number from one to nine which indicates which of the tables the label applies to. There are no operands required on the macro.
   
   3. Each table will be in ascending order based on time in service.
   
   4. Update each entry in all nine tables as follows:
      
      \[
      \text{New Base Pay} = \text{Old Pay} + \text{Old Pay} \times 5.52\% 
      \]

   5. When the tables have been updated print out the detail lines as shown on the Form Layout.

   6. Edit the pay fields with a floating dollar sign, and a decimal pnt.

   7. Use at least one BCT, BXLE, or BXH in your program.

VI. Edit Factors: None.
FOR INSTRUCTIONAL PURPOSES ONLY

EDIT RH DOLU PRINT MOUNTS WITH FLOATING "$"
I. Introduction: You have been interviewed for an ALC programming position by the Southeastern Box Company. They are interested in hiring you but prior to making their decision, they want you to submit a sample program. This company uses binary math for all calculations, so your sample program will demonstrate your ability in the use of binary math instructions. The firm also requires detailed source documentation on all ALC programs so this will be an additional requirement on this sample.

II. Definitions: None.

III. Input/Output:

A. Input: The input will be on cards (see attached card layout). Device address = SYSIPT.

B. Output: The output will be on the 1403 line printer (see attached form layout). Device address = SYSLST.

IV. Assumptions: None.

V. Methods:

A. Print a title and column headings at the top of each page.

B. Using a line counter, print only 20 double spaced detail lines per page.

C. Initialize a field called TOTAL at zero.

D. The indicator field on the input, when valid, will contain a one, two, or space. When it contains a one, bypass all ADD instructions until you encounter a two or a space in the indicator field.

E. The processing code from the input will determine what calculations should be performed on the number read from the card. If the processing code is a --

   one - add the number to TOTAL;

   two - Subtract the number from TOTAL

   three - Divide the number into TOTAL keeping the quotient as the new TOTAL

   four - Multiply the number times TOTAL

   five - Add the number to TOTAL, then Multiply the new TOTAL by the number
six - Divide the number into TOTAL, then subtract the number from the new TOTAL

*** Any processing code that is larger than 6 is invalid.

F. Print out each card with the contents of TOTAL out to the right of the card.

G. If any errors are found on the card, bypass all calculations and print out the card, leaving the TOTAL field blank.

H. If the TOTAL is negative, edit a minus sign on the right side of the TOTAL field on the output.

VI. Edit Factors: Edit the card to insure that all data fields are numeric, except the indicator field which may be a space.
## CARD LAYOUT 10

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>P</th>
<th>I</th>
<th>NOT USED</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>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
</tr>
<tr>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
</tr>
<tr>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
</tr>
<tr>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
</tr>
<tr>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
</tr>
<tr>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
</tr>
<tr>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
</tr>
<tr>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
</tr>
<tr>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
</tr>
</tbody>
</table>

P = Processing Code

I = Indicator
<table>
<thead>
<tr>
<th>Size</th>
<th>Item</th>
<th>Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&quot;</td>
<td>Item1</td>
<td>1</td>
<td>Item Info</td>
</tr>
<tr>
<td>2&quot;</td>
<td>Item2</td>
<td>2</td>
<td>Item Info</td>
</tr>
<tr>
<td>3&quot;</td>
<td>Item3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4&quot;</td>
<td>Item4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5&quot;</td>
<td>Item5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6&quot;</td>
<td>Item6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7&quot;</td>
<td>Item7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8&quot;</td>
<td>Item8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9&quot;</td>
<td>Item9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10&quot;</td>
<td>Item10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11&quot;</td>
<td>Item11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12&quot;</td>
<td>Item12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13&quot;</td>
<td>Item13</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>
You are to write an ALC program that will determine which numbers in a given interval are perfect squares, and which numbers are prime.

(Prime numbers are those numbers that are only divisible by 1 and themselves).

**INPUT:** The program will read one card. This card will contain the upper and lower bounds of the interval. The format of the card is shown on Card Layout 11.

**OUTPUT:** The output should start at the top of a new page. Detail lines should follow the appropriate headings.

Each detail line will consist of 16 8 character fields. Each field will contain one number, right justified. See Form Layout 11.

**Useful Information:**

1. If a number is a perfect square it is not a prime.
2. If a number other than 2 is even it cannot be a prime.
3. If checking a number N to see if it is prime, it is only necessary to check for divisibility by numbers \( \leq \sqrt{N} \).
<table>
<thead>
<tr>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5</td>
<td>5 7 8 8 10 11</td>
</tr>
<tr>
<td>1 2 3 4 5</td>
<td>5 7 8 8 10 11</td>
</tr>
<tr>
<td>6 1 2 3 4</td>
<td>6 1 2 3 4</td>
</tr>
<tr>
<td>6 1 2 3 4</td>
<td>6 1 2 3 4</td>
</tr>
<tr>
<td>6 1 2 3 4</td>
<td>6 1 2 3 4</td>
</tr>
<tr>
<td>6 1 2 3 4</td>
<td>6 1 2 3 4</td>
</tr>
<tr>
<td>6 1 2 3 4</td>
<td>6 1 2 3 4</td>
</tr>
<tr>
<td>6 1 2 3 4</td>
<td>6 1 2 3 4</td>
</tr>
</tbody>
</table>
I. Introduction: The current personnel roster is maintained on disk as a sequential file, but due to additions the file is no longer in the proper order. You are to write an ALC program that will read the disk file, sort the records, and rewrite the records in ascending SSN order.

II. Definitions: None.

III. Input/Output:

A. Input: Your input is on sequential disk in the format shown on the attached disk layout. The filename for your input file is DISKINP and it is at device address SYS005.

B. Output: Your output will be on sequential disk and will be in the same format as it was on the input. The filename for output is DISKOPT and the device address is SYS006.

IV. Assumptions: There will be no more than 30 records on the file.

V. Methods: You may use any strategy you want to sort the records, but you must do your sort in core.

One method of sorting is the 'Bubble Sort'. To use this method first read all records into memory. Then take the first record and compare it to all subsequent records. If you find a SSN lower than the SSN you are using switch the two records and use the new SSN to continue the compares. At the end of the first pass you will have the record with the lowest SSN in the first position. Repeat the process beginning with the second record, then the 3rd and so forth until you have repeated it for all but the last record. At this point all of the records will be in the correct order.

EXAMPLE:

9, 6, 2, 3, 1, 4, 5, 7, 8

We would compare '9' to '6'. Since '6' is less than '9' we switch the '6' and the '9' and continue comparing with the '6'. Continuing with the '6' we find that '2' is less than '6' so once again we switch and continue this time with the '2'. At the end of the first pass the string would look like this: 1, 9, 6, 3, 2, 4, 5, 7, 8.

For the second pass we start with the second record which in this case is the '9' again. We compare it to the '6' (3rd record) switch and continue with the '6' and so forth. At the end of the second pass it would look like this: 1, 2, 9, 6, 3, 4, 5, 7, 8.

VI. Edit Factors: None.

VII. Special Note: The output disk will be dumped for you so that you may check your results. You need do nothing to cause this to happen.
<table>
<thead>
<tr>
<th>ORG</th>
<th>SSN</th>
<th>NAME</th>
<th>PG</th>
<th>RANK</th>
<th>ETS</th>
<th>DATE OF RANK</th>
<th>MON</th>
<th>SER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
<td>11</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>44</td>
<td>44</td>
<td>5</td>
</tr>
</tbody>
</table>

**MISCELLANEOUS INFORMATION**

<table>
<thead>
<tr>
<th>5</th>
<th>8</th>
</tr>
</thead>
</table>

* PAY GRADE
You are to write an ALC program that will generate a bar graph showing the percentage fill for all Federal Stock Numbers.

INPUT: Your program will read the fill data from an unlabeled tape. The tape may contain more than one entry for each FSN. The format of the tape is shown on Tape Layout 13.

The filename for the tape is TAPEIN and the Device address is SYS007.

OUTPUT: The output will consist of a heading at the top of each page and a number of detail lines (one for each FSN). The detail lines will contain three fields: the FSN, the percentage fill, and a row of asterisks representing the percentage fill (one asterisk per percent). See Form Layout 13.

The device address is SYSLST.

Notes: You will have to watch for zero fill items and items which have not been ordered during the period. There is also the possibility that there may be errors on the tape and that the quantity filled will be greater than the quantity ordered. Print zero fill items as part of the graph. List the zero order items on a separate page, one per line under an appropriate heading (See Form Layout 13b). Also on this final page should be one line with the phrase 'LOWEST FILL PERCENTAGE WAS FOR', followed by the FSN of the item with the lowest fill percentage and
the percentage. If there are two with the same percentage the one with the largest quantity ordered should be listed. If there are more of an item listed as filled than were ordered, list the FSN as part of the graph but instead of putting in a percentage put in the word 'ERROR'.
<table>
<thead>
<tr>
<th>FSN</th>
<th>QTY Ordered</th>
<th>QTY Filled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>
You are to write an ALC program to perform edit checks on cards that will be input to a requisition processing run.

INPUT: The format is as shown on Card Layout 14.

The fields should be edited as follows:

- The FSN should be numeric (0-9).
- The nomenclature may contain alphanumeric information (0-9 or A-Z) or blanks or dashes.
- The unit of measure should be alphabetic (A-Z)
- The quantity should be numeric (0-9).
- The activity code should contain alphabetic information in the first two positions and numeric information in the last four.

OUTPUT: The heading is as shown on Form Layout 14.

The detail line should consist of the card image and the name of the column in error. Use FSN for federal stock number and NONE if no errors are found on the card.

The logical name for the printer is SYSLST, double space all output.
<table>
<thead>
<tr>
<th>Federal Stock Number</th>
<th>Nomenclature</th>
<th>UM</th>
<th>Quan</th>
<th>Activity Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9</td>
<td>10 11 12 13 14 15 16 17 18</td>
<td>19</td>
<td>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</td>
<td></td>
</tr>
</tbody>
</table>
UNIVERSITY OF ARMY
INSTITUTE OF ADMINISTRATION
MAY 1978
PERFORMANCE TEST
DEBUGGING A COBOL PROGRAM USING ALC

INSTRUCTIONS:

1. INDIVIDUAL WORK IS MANDATORY. Appropriate disciplinary action will be taken against students who give or receive unauthorized help.

2. Special instructions are listed on page 1 of this booklet.

3. Insure that your class and student number are on the individual grade sheet and/or punchout cards.
Programs which do not go to a normal end of job are canceled by the operating system, and are considered to have abnormally terminated (ABEND). When this happens, page 1 of the program listing will contain, among other items, a system completion code (See Incl 1). The completion code is an indication of what caused the program to ABEND. Following is a list of the most commonly encountered system completion codes, a description of each, and an indication of the probably cause of the ABEND. Those causes marked with an asterisk (*) should be checked first.

**COMPLETION CODE - 001.**

a. **Error Description.**
   
   Input/Output error.

b. **Probable Causes.**
   
   (*) Wrong length record.
   
   (*) Wrong length physical block.

   No end of file marker.
   
   (*) Attempt to read record after end-of-file condition found.

   Physical damage to recording medium.

   Device malfunction.

   (*) Attempt to read from or write to a file which—

   (1) Is closed.

   (2) Was not opened.

**COMPLETION CODE - 013.**

a. **Error Description.**

   Detection of conflicting or unsupported parameters during OPEN processing.

b. **Probable Cause.**

   (*) Check for incorrect external name in a SELECT statement.

**COMPLETION CODE - 031.**

a. **Error Description.**

   An input/output error was detected when processing under the indexed sequential access method (ISAM).
b. **Probable Causes.**

Physical damage to recording medium or device.

(*) Out of sequence key when loading an ISAM data set.

(*) Wrong length record or block.

**COMPLETION CODE - 03B.**

a. **Error Description.**

The error occurred during OPEN processing for an indexed sequential file (ISAM).

b. **Probable Causes.**

(*) The file had not been created.

The file had not been closed after creation.

**COMPLETION CODE - 03D.**

a. **Error Description.**

The error occurred during OPEN processing for an indexed sequential file (ISAM).

b. **Probable Causes.**

Indexed sequential organization not specified.

(*) Check for omission of "ACCESS IS RANDOM" clause in select statement for ISAM file.

**COMPLETION CODE - OC1.**

a. **Error Description.**

The operation code detected is not valid.

b. **Probable Causes.**

A branch to a data area; fetching of data as an operation code for this instruction.

(*) A missing or misspelled external name on SELECT statement.

(*) A file had not been opened when an input/output instruction was issued for it.

A file had been closed when an input/output instruction was issued for it (this may also cause an OC5 termination).
COMPLETION CODE - OC5.

a. Error Description.

An address is specified that is outside of the available storage of the particular computing system.

b. Probable Causes.

Invalid data address.

(*) Indexing (subscripting) outside the program's assigned limits.

(*) Uninitialized index (or subscript). This may also cause a data exception (OC7).

An input/output instruction triggered termination because OPEN was unable to complete its task.

A missing or misspelled external name in SELECT statement.

An attempt to CLOSE a file a second time.

(*) COBOL: An improper exit from a procedure being operated on by a PERFORM statement.

COBOL: A sort operation is being attempted with an incorrect cataloged procedure.

(*) COBOL: An attempt to reference an input/output area before READ or OPEN statement has been issued for the file.

COMPLETION CODE - OC7.

a. Error Description.

Data in a field was of incorrect format for the instruction attempting to process it.

b. Probable Causes.

(*) A data field was not initialized, e.g., blanks were read into a field designed to be processed with packed decimal instructions.

A packed decimal field had an incorrect sign field.

Uninitialized index or subscript. This may also cause an addressing (OC5) or protection (OC4) completion code.

Fields in decimal arithmetic overlap incorrectly.

The decimal multiplicand has too many high-order significant digits.

(*) The index (or subscript) value was incorrect and invalid data was referenced. This could also cause an addressing (OC5) or protection (OC4) completion code.
COBOL: Data was moved from the DISPLAY field to the COMPUTATIONAL or COMPUTATIONAL-3 field at group level. No conversion was performed; invalid data for COMPUTATIONAL-3 (packed decimal) results.

COBOL: The figurative constants ZERO or LOW-VALUE was moved to a group level numeric field.

COBOL: Omission of USAGE clause or inclusion of an erroneous USAGE clause.

FOR USAIA USE: (See Incl 1)

a. Extract Program Interruption Address (PIA) from page 1 of program listing.

b. Determine entry point (EPA) for program from page 1 of program listing. It will be either—
   (1) 01C020.
   (2) 034020.

c. Subtract EPA from PIA. The result is a displacement. Remember, all addresses are in hexadecimal as is the displacement. Now go to the page in the listing which has the heading 'CONDENSED LISTING'. Each verb used in the source program is listed and has associated with it, a statement number and a displacement. To find the statement which caused the program to ABEND, locate the displacement on the condensed listing which most nearly matches and is less than the displacement you just calculated. The corresponding statement number is the one which caused the problem. Now, go back to the probable cause paragraph and correct problem.

COMPLETION CODE - OC9 OR OCB.

a. Error Description.
   Divide exception.

b. Probable Cause.
   Attempting to divide by zero.

COMPLETION CODE - D37.

a. Error Description.
   Space allocated for a data set on a direct access device was exceeded.

b. Probable Cause.
   The program may be in a loop and is writing an infinite number of records to an output file.
COMPLETION CODE - 222.

a. **Error Description.**
   
   Operator canceled the job.

b. **Probable Causes.**
   
   The job may have been canceled because it appeared to be in a loop or because it was waiting for resources that were not immediately available. Perhaps the job was canceled to correct a system interlock condition. There are many reasons why an operator might cancel a job. There may be nothing wrong with your program.

COMPLETION CODE - 322.

a. **Error Description.**
   
   Execution of a program took longer than the time allocated.

b. **Probable Cause.**
   
   Program is in a loop.
//TCELEKES JCE (CCE,GEE,5,6,...N,1)
// 'C:NEIL C:E',CLASS=8,LEVEL=1
IEF1421 - STEP WAS EXECUTED - CCND
IEF3731 STEP /CCB / START 781
IEF3731I STEP /CCB / STCP -
IEF1421 - STEP WAS EXECUTED -
IEF3731I STEP /LKEC / START
IEF3731 STEP /LKEC / STCP
CONTROL BYTE=00 TCB FLAGS=A
 COMPLETION CODE - SYSTEM=OC7 USER=0003
PROGRAM INTERRUPTION (DATA) AT LOCATION 035170
REGISTER SET 1
GFR 0-7 000034EEA 00035176 00034EC4 000355EC
GFR6-15 00004E368 0000F220 0000E55 00036C
INSTRUCTION IMAGE F211C12D 00004F340 0004E005
FPR 0-4 00000000 00000000 00000000 00000000
ACTIVE RE LIST
PROGRAM ID=801C RB TYPE=00 ENTRY POINT=0062FE
RESUME PSW SM=FF K=3 AMMP=5 IC=000D IL+CC=0 PC=5
IEF2241 ALLOCATED FOR TCELEKES CO 008 AT ABEEC
IEF237I 143 ALLOCATED TO PGM=FE70
IEF237I 211 ALLOCATED TO SYSCL1
IEF237I 212 ALLOCATED TO SYSCL2
IEF237I 141 ALLOCATED TO SYSCL6
IEF237I 141 ALLOCATED TO SYSCL7
IEF237I 143 ALLOCATED TO SYSCL8
IEF285I SYST0802.T12344.RFOO0.IELEKES.CASET
IEF285I VCL SER NOS= M4T503K.
IEF285I USAIA.SOFFTWARE.CATASET.012
IEF285I VCL SER NOS= M4T501.
IEF285I USAIA.SOFFTWARE.CATASET.013
IEF285I VCL SER NOS= M4T501.
IEF285I SYST0802.T12344.RF000.IELEKES.00SET
IEF285I VCL SER NOS= M4T503.
IEF273I STEP /GO / START 78082.134
IEF3741 STEP /GO / STCP 78082.1335 CPL
IEF285I SYST0802.T12344.RFOO0.0ELEKES.G0SET
IEF285I VCL SER NOS= M4T503.
IEF375I JOB /TCELEKES / START 78082.1329
IEF376I JOB /TCELEKES / STOP 78082.1335 CPL
SUBTRACT THE ENTRY POINT ADDRESS
FROM THE INTERRUPT ADDRESS:
035170
- 034020
00115C
MIN 03.04SEC MAIN 16K LOS OK
DELETED
USING THE CONDENSED LISTING, DETERMINE THE COBOL STATEMENT CAUSING THE INTERRUPT.
EXIT.

MOVE 'CNN' TO WS-OM-READ-SW.
MOVE 'DELETION' TO WS-DL-STATUS.
PERFORM 0080-WRITE THRU 0080-EXIT.
EXIT.

MOVE IMOI-EM-RCC TO OM 1-NEM-RCD.
MOVE 'DELETION' TO WS-DL-STATUS.
PERFORM 0080-WRITE THRU 0080-EXIT.
EXIT.

MOVE IMOI-EM-RCC TO OM 1-NEM-RCD.
MOVE 'DELETION' TO WS-DL-STATUS.
PERFORM 0080-WRITE THRU 0080-EXIT.
EXIT.

MOVE IMOI-EM-RCC TO OM 1-NEM-RCD.
MOVE 'DELETION' TO WS-DL-STATUS.
PERFORM 0080-WRITE THRU 0080-EXIT.
EXIT.

0080-WRITE.

IF WS-LINE-COUNT GREATER THAN 19
WRITE CR01-PRINT-RCD FRCM WS-READING-LINE-1
AFTER ADVANCING 1 LINE
WRITE CR01-PRINT-RCD FRCM WS-READING-LINE-2
AFTER ADVANCING 2 LINES
MOVE ZER0 TO WS-LINE-COUNT.
MOVE OMOI-SSN1 TO WS-CL-SSN1.
MOVE OMOI-SSN2 TO WS-CL-SSN2.
MOVE OMOI-SSN3 TO WS-CL-SSN3.
MOVE OMOI-NAME TO WS-CL-NAME.
MOVE OMOI-INS TO WS-CL-INS.
MOVE OMOI-STAT-ADD TO WS-CL-STAT-ADD.
MOVE OMOI-CITY TO WS-CL-CITY.
MOVE OMOI-JC-CODE TO WS-CL-JC-CODE.
IF OMOI-PAY NUMERIC
MOVE OMOI-PAY TO WS-CL-PAY.
IF OMOI-FICA NUMERIC
MOVE OMOI-FICA TO WS-CL-FICA.
IF OMOI-IRS NUMERIC
MOVE OMOI-IRS TO WS-CL-IRS.
WRITE CR01-PRINT-RCD FROM WS-DETAIL-LINE
AFTER ADVANCING 2 LINES.

POSSIBLE CAUSE: WS-LINE-COUNT IS NOT INITIALIZED. CHECK DATA DIVISION.
FD OM-NEM-FILE
BLOCK CONTAINS 5 RECORDS,
RECORD CONTAINS 80 CHARACTERS,
LABEL RECORDS ARE STANDARD,
DATA RECORD IS OM01-NEM-RCC.

01 OM 1-NEM-RCC.
  05 GM01-SSN.
     10 GM01-SSN1
     10 GM01-SSN2
     10 GM01-SSN3
  05 GM01-NAME
  05 GM01-IN
  05 GM01-SIT-ADD
  05 GM01-CITY
  05 GM01-JC-CODE
  05 GM01-PAY
  05 GM01-FICA
  05 GM01-IRS
  05 FILLER

WORKING-STORAGE SECTION.
77 WS-TRANS-READ-SW
VALUE 'ONN'.
  05 FILLER
  05 FILLER
  05 FILLER
  05 FILLER
VALUE SPACE.
77 WS-CM-READ-SW
VALUE 'ONN'.
  05 FILLER
  05 FILLER
  05 FILLER
  05 FILLER
VALUE 'OFF'.
77 WS-READ-COM-SW
77 WS-LINE-COUNT
77 WS-NINES
VALUE SS55555555.

01 WS-READING-LINE-1.
  05 FILLER
VALUE SPACE.
  05 WS-PDI-TITLE
VALUE 'DAC MASTER UPDATE REPORT'.
  05 FILLER
VALUE SPACE.

01 WS-READING-LINE-1.
  05 FILLER
VALUE SPACE.
  05 FILLER
VALUE 'STATES'.
  05 FILLER
VALUE 'SOC-SEC-NUM'.
  05 FILLER
VALUE SPACE.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1-1</td>
</tr>
<tr>
<td>Disk Operating System (DOS)</td>
<td>1-4.5</td>
</tr>
<tr>
<td>Initial Program Load (IPL)</td>
<td>1-5</td>
</tr>
<tr>
<td>Multiprogramming</td>
<td>1-5.4</td>
</tr>
<tr>
<td>Job Control Language</td>
<td>3-1</td>
</tr>
<tr>
<td>Compile/linkedit/execute</td>
<td>3-4</td>
</tr>
<tr>
<td>JOB card</td>
<td>3-5</td>
</tr>
<tr>
<td>OPTION card</td>
<td>3-5</td>
</tr>
<tr>
<td>USAIA standard options</td>
<td>3-10</td>
</tr>
<tr>
<td>EXEC card</td>
<td>3-11</td>
</tr>
<tr>
<td>Linkage Editor</td>
<td>4-1</td>
</tr>
<tr>
<td>PHASE card</td>
<td>4-1</td>
</tr>
<tr>
<td>INCLUDE card</td>
<td>4-2</td>
</tr>
<tr>
<td>ENTRY card</td>
<td>4-2</td>
</tr>
<tr>
<td>ACTION card</td>
<td>4-3</td>
</tr>
<tr>
<td>Linkedit map</td>
<td>4-3</td>
</tr>
<tr>
<td>Assignment of I/O devices</td>
<td>5-1</td>
</tr>
<tr>
<td>ASSGN card</td>
<td>5-4</td>
</tr>
<tr>
<td>ADAM</td>
<td>5-4.2</td>
</tr>
<tr>
<td>Data file specification</td>
<td>6-1</td>
</tr>
<tr>
<td>Tape labels</td>
<td>6-1</td>
</tr>
<tr>
<td>TLBL card</td>
<td>6-3</td>
</tr>
<tr>
<td>LBLTYP card</td>
<td>6-5</td>
</tr>
<tr>
<td>Disk files</td>
<td>7-1</td>
</tr>
<tr>
<td>DLBL card</td>
<td>7-2</td>
</tr>
<tr>
<td>EXTENT card</td>
<td>7-2</td>
</tr>
<tr>
<td>COBOL PE2/PE3 JCL exercise</td>
<td>7-7</td>
</tr>
<tr>
<td>Indexed Sequential (ISAM) files</td>
<td>7-8</td>
</tr>
<tr>
<td>Librarian</td>
<td>8-1</td>
</tr>
<tr>
<td>Service (list/punch) functions</td>
<td>8-3</td>
</tr>
<tr>
<td>Catalog functions</td>
<td>8-13</td>
</tr>
<tr>
<td>Rename Functions</td>
<td>8-17</td>
</tr>
<tr>
<td>Delete Functions</td>
<td>8-18</td>
</tr>
<tr>
<td>Condense functions</td>
<td>8-20</td>
</tr>
<tr>
<td>Update functions</td>
<td>8-21</td>
</tr>
<tr>
<td>Reallocate functions</td>
<td>8-25</td>
</tr>
<tr>
<td>Copy functions</td>
<td>8-26</td>
</tr>
<tr>
<td>DOS utilities</td>
<td>9-1</td>
</tr>
<tr>
<td>File-To-File Utilities</td>
<td>9-5</td>
</tr>
<tr>
<td>Utility control cards</td>
<td>9-7</td>
</tr>
<tr>
<td>TABLE OF CONTENTS (CONT)</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>Operating Systems (OS)</td>
<td>10-1</td>
</tr>
<tr>
<td>APPENDICIES:</td>
<td></td>
</tr>
<tr>
<td>DOS Job Control flow chart</td>
<td>A</td>
</tr>
<tr>
<td>Compile/Linkedit/Execute diagram</td>
<td>B</td>
</tr>
</tbody>
</table>
HISTORY OF COMPUTERS:

The first type of computing machines that were used were the punch card machines. These machines used plugboards to store their programs. They were comparatively slow, had very limited capabilities and had no internal storage.

The PCM were followed by the first generation computers. These were big, relatively slow vacuum tube machines. All programming for these machines was done at the machine language level. They were single job initiation machines and were used mainly as scientific calculators.

About 1955-56, the second generation machines emerged. These had transistors instead of tubes and were faster than their first generation predecessors. It was during the second generation that compiler level languages such as COBOL and FORTRAN were developed. The second generation machines had a batch job capability and were the first of the general purpose machines. They were now used for Business Data Processing as well as scientific calculators.

Time marches on...about 1961, enter the third generation machines replacing the transistor with semiconductor circuits once again increasing the speed. Magnetic core was also being produced relatively cheaply so these computers had large core memories.

All of these changes did not come without some problems, most of which started in the second generation. The hardware technology was ahead of the software support so there was an inefficient use of the hardware. Large computers were not effective. Many of their resources such as memory and I/O devices sat idle a lot of the time. There was too much human intervention; too much interaction between man and machine. Even though there was too much interaction between man and machine, there was no way for man to interact with man thru the machine, in other words, there was no way to link one program to another.
The solution to these problems was an operating system. An operating system is an integrated set of programs designed to improve overall operating effectiveness. The purpose of an operating system is to increase the amount of data processed in a given amount of time. An operating system will not reduce the execution time for a single program. Over the years there have been many operating systems. One of the first was the Share Operating System (SOS) written by a group of IBM users. The SOS did not pan out. IBM produced IBSYS for its 7090 line of computers. This was about 1959. In 1961, Burroughs introduced the Master Control Program (MCP). It was the first of the third generation operating systems. It was not, however, until about 1963 when IBM announced DOS and OS that operating systems were really here to stay. There are three basic sources for operating systems. The first and least common is the user group, the time and expense (about $60,000,000 for DOS) makes it almost impossible for a group of users to produce their own system. The second source is the Academic World. Universities often write their own special application systems using the wealth of research personnel on their faculties and any free graduate students. The last and most common source of operating system is the hardware manufacturer. Almost all operating systems come from this last source. There are three basic requirements for an operating system. The first is a control program to control all the systems' resources. The second is a language for man-system communication and finally a software library and some means for retrieving programs from that library. Not all operating systems will have all of these requirements in a very obvious form but these three needs will in some manner be present in every operating system.

Reduction in Costs: One of the first computers designed and built costs $500,000, had 18,000 vacuum tubes, weighted 30 tons and was named "ENIAC". Today, a microprocessor can have the same computing power of ENIAC, can be the size of a baby's fingernail and cost $10.
TYPES OF OPERATING SYSTEMS:

BPS (Basic Programming Support) - a minimum system resident in cards or tape
for use in installations having no disk files. Support Assembler Language and RPG.

BOS (Basic Operating System) - A disk resident system supporting Assembler
Language and RPG, with sufficient capacity to satisfy requirements of small installations.

TOS (Tape Operating System) - A tape resident system offering the capabilities
of supporting all languages plus multiprogramming, storage protection, and the internal
timer with no disk.

DOS (Disk Operating System) - a disk resident system for users requiring more
advanced operational support than that of TOS. Supports all languages as well as remote
processing, storage protection, multiprogramming, and the interval timer.

OS (Operating system) - a disk resident system that provides for the assignment
of devices and auxiliary storage space at execution time. Provides for multiprogramming
fifteen separate jobs at one time. Also provides an internal SPOOLing capability as well
as a priority scheduling system.

OS/VS (Operating System / Virtual Storage) - a disk resident system that uses a
Direct access storage device to simulate a larger memory capacity than the machine really
has. For example a System 370/165 with 512,000 bytes of real memory can be made to per-
form as though it has 2,000,000 bytes of real memory.

OPERATING SYSTEM.

An Operating System - intergrated set of programs designed to improve the total
operating efficiency of a computer's operation.

Functions of the Operating System.

Scheduling of I/O operations

Protection of partition boundaries (required for multiprogramming)

Error handling as referenced to the five interrupts.
Machine check interrupt
Supervisor call (SVC) interrupt
Program check interrupt
External interrupt
I/O interrupt

Operator communication with CPU.

Provide for program loading from private or system core image library.

Provide for program termination.

REQUIREMENTS OF THE OPERATING SYSTEM.

A control program (supervisor)
Library facilities
  Core image library
  Source statement library
  Relocatable library.

A device which provides a means for communication between the operating system and the computer operator (console typewriter).

NOTE: These libraries will be present in the operating system in some form.
DISK OPERATING SYSTEM

CONTROL PROGRAMS
- IPL
- SUPERVISOR
- JOB CONTROL
- LINKAGE EDITOR

SERVICE PROGRAMS
- LIBRARIAN
- LANGUAGE TRANSLATORS
  - COBOL
  - FORTRAN
  - ALC
  - BASIC
  - PL/1
  - ETC.

PROCESSING PROGRAMS
- UTILITIES
- PROCESSING PROGRAMS
  - LOCAL UNIQUES
  - SIDPERS
  - SAILS
  - STANFINS
  - ETC.

MAINT O

DIAGRAM: DISK OPERATING SYSTEM

Diagram shows the hierarchical structure of the disk operating system, with control programs, service programs, and processing programs.
V. Types of programs which make-up the operating system.

a. Initial Program Load (IPL)

INITIAL PROGRAM LOAD

IPL - is loaded by the operator dialing the address of SYS RES on the CPU and pressing the console load key. The loading of IPL is a machine function, no instructions are executed. After the IPL is loaded, it is given control of the machine. At this time IPL clears main memory and the registers, loads the supervisor in the low core of main memory and passes control to the supervisor. When these tasks are accomplished, IPL has no further usage.
The time when the supervisor is given control of the machine, the computer operator sets the date and time of day to be used by the supervisor and Problem Programs. He may also add or change the I/O device assignments in the Logical Unit Block (LUB) and Physical Unit Block (PUB) of the supervisor. Finally, he can change the partition sizes of the partitions allocated to the machine by entering an ALLOC statement.

**EXAMPLE:**

- OIOA Give IPL Control commands
- SET DATE = DD/MM/YY, clock=HH/MM/SS
- OI201 DOS IPL Complete

At this point control is given to the control program (Supervisor).

The supervisor loads the Job Control Program into the Background partition. Job control finds the standard assignment for SYSRDR in the LUB and PUB. Next it reads the job control statements from the card reader (SYSRDR) performs the following functions:

1. Assigns device addresses to symbolic units.
2. Set up fields in the communication region.
3. Edit and stores label information.
4. Prepares for restarting checkpoints programs.
5. Clear the problem area to binary zeros between job steps.
6. Finally, it passes the name of the program to be executed to the System Loader.

7. Control is then passed to the Supervisor.

When the supervisor is given control from the Job Control Program, the System Loader goes to System Residence Pack (SYSRES), Core Image Library and fetches the program to be executed by placing it into main memory. Next the address of the first executable instruction is stored in the Control Unit of the CPU.
Finally, the problem program is given control and begins execution. The preceding process explains what happens in the computer to start a problem program executing in the Background (BG) partition.

In order to start jobs in the Foreground 1 (F1) and Foreground 2 (F2) partitions, the operator must press the request key on the console typewriter and enter the following statements:

```
AR    BATCH F1
F1    ASSGN SYSXXX,X'CUU'
PRESS THE REQUEST KEY
AR    BATCH F2
F2    ASSGN SYSXXX,X'CUU'
```

This process caused the Job Control Program to be loaded in the F1 and F2 partitions. Once Job Control is given control, it repeats the steps indicated above which cause the problem program to begin executing in each partition.

The Job Control Program is written over in the partitions by the problem program being loaded into the partition by the System Loader. When the problem program ends execution (Stop Run), control is given to the Supervisor. The supervisor reloads the Job Control Program into the partition of the program that ended execution wiping out the program. Job Control then reads SYSRDR to obtain the Job Control statements for the next task to be performed. This process continues until all jobs in the input have been processed through the computer.
JOB LOADING AND EXECUTING

MAIN MEMORY

SUPERVISOR NUCLEUS

SYSLDR

Physical Transient Area

Logical Transient Area

JOB CONTROL

BG

F2

F1

SYSRES

IPL VOL

Directory

CORE IMAGE LIBRARY

Directory

RELOCATABLE LIBRARY

Directory

SOURCE STATEMENT LIBRARY

1-5.3

CSD.SFW.OS
MULTIPROGRAMMING - concurrent execution of two or more programs simultaneously residing in core storage of a computer.
b. Control Program is the backbone of the operating system and is composed of the following components:

1. Job Management - identifies the job to be done and assigns I/O devices for the problem program.

2. Task Management (supervisor) brings in the problem program from the library and responds to any error conditions by unloading one program and going on to the next one.

c. Data Management - handles movement of data between main storage and I/O devices. This also includes:

1. Scheduling and programming the operation of channels.

2. The identifications of data by volume (the device in which data is stored such as a reel of tape, or a disk pack)

3. The resolution of error conditions that occur during the movement of data.

4. Data Management is composed of the Input Output Control System, (referred to as IOCS)

   a. IOCS is a set of routines (programs) that handle the movement of data between main storage and I/O devices. These IOCS routines fall into two categories:

   I. Physical IOCS (PIOCS) - is composed of those I/O routines which supervise the reading and writing of data on I/O devices without regard for its logical content, format or organization.

   II. Logical IOCS (LIOCS) - Once the data has been brought from the I/O device by PIOCS, the LIOCS routines now operate on this data to make its logical content available to the problem program as required.
(b.) File Access Methods.

(I) Sequential access method - sequential processing is used to read/write and process successive records in a logical file.

(II) Index sequential access method - an index sequential file is built from sequential input and has indexes that permit individual records to be found for subsequent processing operations. By supplying the key of any record, a programmer can obtain the specific record randomly.

(III) Direct access method - records are usually organized in a random manner and are processed by referring to a record location reference supplied in the problem program. This location reference indicates the exact record by track and key or identifier.

(IV) Basic Teleprocessing Access Method (BTAM) controls transmission and reception of messages over the communication lines in response to READ and WRITE macros. The LIOCS technique of accessing the file is not extended to the problem program.

(V) Queued Teleprocessing Access Method - has the same capabilities as BTAM with the addition of LIOCS technique of accessing the file by the problem program.

(c) Processing Programs.

(I) Language processor - are programs which use as input source programs and interpret them into a machine compatible language.

(a) FORTRAN compiler
(b) RPG compiler
(c) COBOL compiler
(d) PL/I compiler
(e) ALC assembler
(f) GPSS compiler
(d) Utility programs - generalized programs which, through the use of a control card, can perform a specific function.

(I) Card to tape utility
(II) Tape to print utility
(III) Disk to print utility
(VI) Sort/Merge ...etc.
(V) Autotest.

(e) Service programs - generate the system, create and maintain the library sections, and edit programs into disk residence before execution.

(I) Link Editor Program - prepares programs for entry into the core image library by:
   (a) Assigning final main storage addresses to data and instructions.
   (b) Incorporating subroutines from the relocatable library, as requested by the programmer and combining separately written sections of the program into a single unified program.

(II) Librarian - is a group of programs that maintains and reorganizes the disk library areas and provides printed and punched output from the libraries. Three libraries are used.
   (a) Core Image Library. All programs in the system (IBM - supplied and user programs) are in machine code and loaded from this library by the system loader routine of the supervisor.
   (b) Relocatable Library: This library stores object modules that can be used for subsequent linkage with other programs modules. A complete program of one or more modules can be placed in this library.
   (c) Source Statement Library. This library stores IBM-supplied MACRO definitions and user defined source statement routines (such as MACRO definitions) built to provide extended program-assembly capability.
(III) Librarian Programs:

MAINT
CSERV
SSERV
RSERV
CORGZ
LISTVTOC

DSERV
## DOS - Main Storage Organization

### Permanent Routines

- **Minimum** 6144 Bytes
- **Supervisor**
  - Permanent Routines
    - Permanent Storage - Locations
      - PSW's, CSW, CAW, etc.
      - I/O unit, control tables
        - LUB, PUB, JIB
      - Communications Region - 46 Bytes
        - Channel
        - EXCP Routing
        - Scheduler
        - I/O Interrupt RTE
        - Start I/O RTE
      - Other
        - SVC
        - Program Check
        - Machine Check
        - External
      - System Loader (SYSLDR)
      - Resident (2311) Error Recovery Routines
      - Storage Protection (Optional)
      - Timer Services (Optional)
- **Transient Routines**
  - Open, close, dump,
    - Checkpoint, operator communications,
      - Error Processing RTES
- **Problem:** User Program
- **Program**
  - Job Control
- **Area**
  - Linkage Editor
  - Librarian Routine
- **(Minimum 10K Bytes)**
  - Language Translator
The Communication Region

DATE

ADDRESS:
PROBLEM PROGRAM LABEL AREA

END OF SUPVR STORAGE, PROTECTION

USER AREA

PROGRAM SWITCHES (UPSI)

JOB NAME

ADDRESS:
UPPERMOST BYTE
OF PROBLEM PROGRAM AREA

ADD-F-SS:
UPPERMOST BYTE
OF PROBLEM PROGRAM PHASE

ADDRESS:
UPPERMOST BYTE
OF LONGEST PROBLEM PROGRAM PHASE

LENGTH OF
PROBLEM PROGRAM LABEL AREA
Diagram of PHYSICAL UNIT BLOCK (PUB) - Where the machine address of the I/O device can be found:

<table>
<thead>
<tr>
<th>CHANNEL</th>
<th>BYTE 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIT</td>
<td>BYTE 1</td>
</tr>
<tr>
<td>POINTER TO CHANNEL QUEUE</td>
<td>BYTE 2</td>
</tr>
<tr>
<td>ERROR RETRY COUNTER OR POINTER TO TAPE ERROR BLOCK (BOTH TEB AND TEBV)</td>
<td>BYTE 3</td>
</tr>
<tr>
<td>DEVICE TYPE</td>
<td>BYTE 4</td>
</tr>
<tr>
<td>DEVICE OPTIONS (TAPE SET MODE, ETC)</td>
<td>BYTE 5</td>
</tr>
<tr>
<td>CHANNEL SCHEDULER FLAGS</td>
<td>BYTE 6</td>
</tr>
<tr>
<td>JOB CONTROL FLAGS</td>
<td>BYTE 8</td>
</tr>
</tbody>
</table>

LOGICAL UNIT BLOCK (LUB) - Where system and programmer logical units are defined:

<table>
<thead>
<tr>
<th>Logical Unit</th>
<th>BYTE 0</th>
<th>BYTE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>*SYSRDR</td>
<td>Pointer to</td>
<td>Pointer to</td>
</tr>
<tr>
<td>*SYSIPT</td>
<td>Pub</td>
<td>Job Information</td>
</tr>
<tr>
<td>*SYSLST</td>
<td>Block</td>
<td></td>
</tr>
<tr>
<td>*SYSLOG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSLNK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSRES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSSLB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSRLB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSCLB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*SYSPCH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSSO00 - SYSSMAX</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* May be used in DTF of ALC
JOB INFORMATION BLOCK (JIB)

The JIB contains one of the following:

1. LUE entry of the standard assignment when a temporary LUB assignment is made.
2. PUB pointer for an alternate LUB assignment.
3. Extent information when DASD files protection is selected as a supervisor generation option.

---

**JIB TABLE**

- JIB 1
- JIB 2
- JIB 3
- JIB 4
- JIB 5
- JIB 6

**Columns:**

- **BYTE 0**: Stored Standard assignment
- **BYTE 1**: LUB Entry of Stored Standard Assignment (PUB & LUB Pointer)
- **BYTE 2**: FLAG TYPE
- **BYTE 3**: Chain BYTE
  - Contains the Displacement Index of Next JIB.
  - 'FF' defines end of Chain
JOB CONTROL LANGUAGE (JCL)

Whereas COBOL is a language for man-machine communication, JCL is a language for man-operating system communication.

JCL can enter the system through many channels. JCL cards may be keypunched by the programmer and placed into the system through the card reader along with such items as a COBOL source deck, a punched object deck, data cards or any combination of these. JCL may be cataloged in the Source Statement library to be invoked by SPOOLing programs. JCL may be entered into the system by the operator thru the computer console device.
STAGES OF PROGRAM DEVELOPMENT

The term program is used to describe several things. The programmer codes sets of source statements that may be a complete program or part of a program. These source statements are then compiled into a relocatable machine-language program which, in turn, must be edited into an executable program, and may be combined with other programs. Consequently, it is convenient to refer to each stage of program development by a particular name.

A set of source statements that is processed by a language translator (Assembler, COBOL, FORTRAN, RPG or PL/I), is referred to as a source module or source deck (if in card form).

The output of a language translator is referred to as an object module (or deck). All object modules must be further processed by the linkage editor before they can be executed in the system.

The output of the linkage editor consists of one or more program phases in the core-image library. A phase is in executable, nonrelocatable, core image form.
THREE PROGRAM FORMS

SOURCE MODULE

COMPILER

RELOCATABLE OBJECT MODULE

LINKAGE EDITOR

LOAD MODULE (PHASE)
The following represents the JCL for a very simple program that reads cards, does some sort of calculations, and prints out a report. The cards bounded by the //JOB and the //& represent what is called a job. This particular job consists of 3 job steps each of which has a //EXEC card associated with it.
The JOB statement indicates the beginning of control information for a job.

**FORMAT**

```
// JOB jobname [accounting information]
```

**jobname** The name of the job. Must be 1 to 8 alphameric characters the first of which must be alphabetic.

**accounting information** If the job accounting interface has been specified during system generation, 16 characters of user specified accounting information can be entered in the job statement. It must be separated from the jobname by a blank.

**// OPTION LIST,LOG,LINK,ERRS**

The OPTION statement specifies one or more of the Job Control options.

**FORMAT**

```
// OPTION option 1, option 2,.....
```

The purpose of this card is to override standard system options set at generation time. Selected options can be in any order. Options are reset to the system standards at the end of each job. (INDICATED BY & CARD)

- **LOG** List all JCL on the printer from this card until a NOLOG, // JOB, or /& is encountered.
- **NOLOG** Stop listing JCL.
- **DUMP** Causes a dump of registers and main storage to be printed on printer if the program abnormally terminates.
- **NODUMP** Turn off the dump option.
LINK  Tells the COBOL compiler to put the object module in the
linkage editor work area known as SYSLNK.

NOLINK  Turn off link option.

DECK  Tells the COBOL compiler to punch an object deck.

NODECK  Turn off the deck option.

LIST  Tells the COBOL compiler to write the source listing
on PRINTER

NOLIST  Turn off the list option.

PROCEDURE DIVISION
OPEN INPUT DISK-FILE
OUTPUT PRINTER.
PERFORM 0010-READ-AND-PRINT THRU 0010-EXIT
UNTIL WS-DISK-EOF-SWITCH EQUAL TO '1'.
MOVE WS-DISK-RCD-COUNT TO WS-TL-DISK-RCD-COUNT.
WRITE QRU1-PRINTER-RCD FROM WS-TOTAL-LINE
AFTER ADVANCING 3 LINES.
CLOSE DISK-FILE
OUTPUT PRINTER.
STOP RUN.

PERFORM 0010-READ-AND-PRINT.
READ DISK-FILE
AT END
MOVE '1' TO WS-DISK-EOF-SWITCH
LISTX  Tells the COBOL compiler to write a PROCEDURE DIVISION MAP on SYSLST.

NOLISTX  Turn off listx option.

86  000726  START  EQU  *
    000726  41 10 C 045  LA  1,045(0,12)  LI+13
    00072A  58 00 D 1C8  L  0,1C8(0,13)  DTF=1
    00072E  18 40  LR  4,0
    000730  07 00  BCR  0,0
    000732  05 F0  BALR  15,0
    000734  50 00 F 006  ST  0,006(0,15)
    000738  45 00 F 00C  BAL  0,00C(0,15)
    00073C  00000000  DC  X'00000000'
    000740  0A 02  SVC  2
    000742  41 10 C 045  LA  1,045(0,12)  LI+13
    000746  58 00 D 1C4  L  0,1C4(0,13)  DTF=2
    00074A  18 40  LR  4,0
    00074C  07 00  BCR  0,0
    000750  05 F0  BALR  15,0
    000754  50 00 F 006  ST  0,006(0,15)
    000758  45 00 F 00C  BAL  0,00C(0,15)
    00075C  00000000  DC  X'00000000'
    000760  0A 02  SVC  2
    000762  45 20 D 1C0  ST  2,1C0(0,13)  BL=2
    000766  58 00 D 1C4  L  8,1C4(0,13)  BL=2
    00076A  50 00 D 1E4  L  0,1E4(0,13)  VN=01
    00076E  58 00 C 010  ST  0,1E4(0,13)  PSV=1
    000772  50 00 D 1E8  L  0,010(0,12)  GN=01
    000776  50 00 D 1E8  ST  0,1E8(0,13)  VN=01
    00077A  05 00  EQU  *
    00077C  00000000  DC  X'00000000'
    000780  07 82  CLC  007(1,6),04D(12)  DNM=1-398
    000784  50 00 D 004  BCR  8,2
    000789  07 00  EQU  *
    00078E  07 F1  L  1,004(0,12)  PN=01
    000792  58 00 D 1E4  BCR  15,1
    000796  07 00  EQU  *
    00079C  58 00 D 1E4  L  0,1E4(0,13)  PSV=1
### SYM

Tells the COBOL compiler to write a DATA DIVISION MAP on the printer.

### NOSYM

Turn off SYM option.

<table>
<thead>
<tr>
<th>Internal Name</th>
<th>Level</th>
<th>Source Name</th>
<th>Base</th>
<th>Displacement</th>
<th>Definition</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNM=1-074</td>
<td>FD</td>
<td>DISK-FILE</td>
<td>DTF=01</td>
<td>000</td>
<td>OCL80</td>
<td>GROUP</td>
</tr>
<tr>
<td>DNM=1-107</td>
<td>01</td>
<td>IM01-DISK-RCD</td>
<td>BL=1</td>
<td>000</td>
<td>DS 3C</td>
<td>DISP</td>
</tr>
<tr>
<td>DNM=1-133</td>
<td>02</td>
<td>FILLER</td>
<td>BL=1</td>
<td>003</td>
<td>DS 18C</td>
<td>DISP</td>
</tr>
<tr>
<td>DNM=1-149</td>
<td>02</td>
<td>IM01-NAME</td>
<td>BL=1</td>
<td>015</td>
<td>DS 0CL9</td>
<td>GROUP</td>
</tr>
<tr>
<td>DNM=1-168</td>
<td>03</td>
<td>IM01-SOC-SEC-NU</td>
<td>BL=1</td>
<td>015</td>
<td>DS 4C</td>
<td>DISP</td>
</tr>
<tr>
<td>DNM=1-197</td>
<td>03</td>
<td>IM01-SSN-PART-1</td>
<td>BL=1</td>
<td>018</td>
<td>DS 2C</td>
<td>DISP</td>
</tr>
<tr>
<td>DNM=1-222</td>
<td>03</td>
<td>IM01-SSN-PART-2</td>
<td>BL=1</td>
<td>01A</td>
<td>DS 4C</td>
<td>DISP</td>
</tr>
<tr>
<td>DNM=1-247</td>
<td>02</td>
<td>FILLER</td>
<td>BL=1</td>
<td>01E</td>
<td>DS 50C</td>
<td>DISP</td>
</tr>
<tr>
<td>DNM=1-291</td>
<td>FD</td>
<td>PRINTER</td>
<td>DTF=02</td>
<td>000</td>
<td>OCL80</td>
<td>GROUP</td>
</tr>
<tr>
<td>DNM=1-322</td>
<td>01</td>
<td>OR01-PRINTER-RCD</td>
<td>BL=2</td>
<td>000</td>
<td>DS 133C</td>
<td>DISP</td>
</tr>
<tr>
<td>DNM=1-348</td>
<td>77</td>
<td>WS-DISK-RCD-COUNT</td>
<td>BL=3</td>
<td>004</td>
<td>DS 10C</td>
<td>DISP-NM</td>
</tr>
<tr>
<td>DNM=1-375</td>
<td>77</td>
<td>WS-LINE-COUNT</td>
<td>BL=3</td>
<td>007</td>
<td>DS 1C</td>
<td>DISP</td>
</tr>
<tr>
<td>DNM=1-398</td>
<td>77</td>
<td>WS-DISK-EOF-SWITCH</td>
<td>BL=3</td>
<td>008</td>
<td>DS 10C</td>
<td>DISP</td>
</tr>
<tr>
<td>DNM=1-426</td>
<td>01</td>
<td>WS-DETAIL-LINE</td>
<td>BL=3</td>
<td>009</td>
<td>DS 0CL132</td>
<td>GROUP</td>
</tr>
<tr>
<td>DNM=1-453</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>00A</td>
<td>DS 10C</td>
<td>DISP</td>
</tr>
<tr>
<td>DNM=1-472</td>
<td>02</td>
<td>WS-NAME</td>
<td>BL=3</td>
<td>00B</td>
<td>DS 10C</td>
<td>DISP</td>
</tr>
<tr>
<td>DNM=1-489</td>
<td>02</td>
<td>FILLER</td>
<td>BL=3</td>
<td>00C</td>
<td>DS 10C</td>
<td>DISP</td>
</tr>
<tr>
<td>DNM=2-000</td>
<td>02</td>
<td>WS-SSN</td>
<td>BL=3</td>
<td>00E</td>
<td>DS 10C</td>
<td>GROUP</td>
</tr>
</tbody>
</table>
XREF  Tells the COBOL compiler to write the symbolic cross reference list on the printer.

NOXREF  Turn off XREF option

**CROSS-REFERENCE DICTIONARY**

<table>
<thead>
<tr>
<th>DATA NAMES</th>
<th>DEFN</th>
<th>REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK-FILE</td>
<td>00025</td>
<td>00096</td>
</tr>
<tr>
<td>IM01-NAME</td>
<td>00038</td>
<td>00109</td>
</tr>
<tr>
<td>IM01-SSN-PART-1</td>
<td>00040</td>
<td>00106</td>
</tr>
<tr>
<td>IM01-SSN-PART-2</td>
<td>00041</td>
<td>00107</td>
</tr>
<tr>
<td>IM01-SSN-PART-3</td>
<td>00042</td>
<td>00108</td>
</tr>
<tr>
<td>PRINTER</td>
<td>00027</td>
<td>00086</td>
</tr>
<tr>
<td>OR01-PRINTER-RCD</td>
<td>00048</td>
<td>00099</td>
</tr>
<tr>
<td>WS-DISK-RSD-COUNT</td>
<td>00051</td>
<td>00112</td>
</tr>
<tr>
<td>WS-DISC-EOF-SWITCH</td>
<td>00053</td>
<td>00110</td>
</tr>
<tr>
<td>WS-NAME</td>
<td>00056</td>
<td>00109</td>
</tr>
<tr>
<td>WS-SSN-PART-1</td>
<td>00060</td>
<td>00106</td>
</tr>
<tr>
<td>WS-SSN-PART-2</td>
<td>00063</td>
<td>00107</td>
</tr>
<tr>
<td>WS-SSN-PART-3</td>
<td>00066</td>
<td>00108</td>
</tr>
<tr>
<td>WS-TOTAL-LINE</td>
<td>00069</td>
<td>00091</td>
</tr>
<tr>
<td>WS-TL-DISK-RCD-COUNT</td>
<td>00072</td>
<td>00090</td>
</tr>
<tr>
<td>WS-PAGE-HEADING</td>
<td>00076</td>
<td>00103</td>
</tr>
</tbody>
</table>

**PROCEDURE NAMES**

<table>
<thead>
<tr>
<th>PROCEDURE NAMES</th>
<th>DEFN</th>
<th>REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010-READ-AND-PRINT</td>
<td>00096</td>
<td>00088</td>
</tr>
<tr>
<td>0010-EXIT</td>
<td>00113</td>
<td>00088</td>
</tr>
</tbody>
</table>
ERRS  Tell the COBOL compiler to summarize all errors in the source program on the printer.

NOERRS Turn off the ERRS option.

---

CARD  ERROR MESSAGE

1  ILA1087I-W ' ON ' SHOULD NOT BEGIN IN AREA A.
1  ILA1087I-W ' LOG ' SHOULD NOT BEGIN IN AREA A.
1  ILA1129I-C ID DIV. HEADER MISSING OR MISPLACED. ASSUMED PRESENT.
1  ILA1004I-E INVALID WORD ON . SKIPPING TO NEXT RECOGNIZABLE WORD.
1  ILA1120I-W COMMA NOT FOLLOWED BY SPACE. ASSUMED.
1  ILA1120I-W COMMA NOT FOLLOWED BY SPACE. ASSUMED.
1  ILA1120I-W COMMA NOT FOLLOWED BY SPACE. ASSUMED.
1  ILA1120I-W COMMA NOT FOLLOWED BY SPACE. ASSUMED.
1  ILA1120I-W COMMA NOT FOLLOWED BY SPACE. ASSUMED.
51 ILA1132I-E INVALID SYSTEM-NAME. SKIPPING TO NEXT CLAUSE.
53 ILA1132I-E INVALID SYSTEM-NAME. SKIPPING TO NEXT CLAUSE.
60 ILA1056I-E FILE-NAME NOT DEFINED IN A SELECT. DESCRIPTION IGNORED.
73 ILA1056I-E FILE-NAME NOT DEFINED IN A SELECT. DESCRIPTION IGNORED.

---

CATAL  Causes the program being link edited to be cataloged into the core-image library.

The Standard System Options of Student jobs are as follows: NOLOG, NODUMP, NOLINK, NODECK, LIST, NOLISTX, NOSYM, ERRS, XREF.
The EXEC statement indicates the end of control information for a job step and that execution of a program is to begin.

**FORMAT**

```
// EXEC progname
```

*Progname* is the name of the program cataloged in the core image library.

What occurs here is that the progname is passed to the supervisor which goes to the core-image library directory, finds the program, loads it into memory, and passes control to it to begin executing.

**NOTE:** FCOBOL is the name of a COBOL compiler.

```
source deck
```

This is the COBOL source deck that is to be compiled.

```
/*
This delimiter card must be the last statement of each input data file. It causes the system to return an end-of-file indication to the program utilizing the data file.
```
This execute statement causes the linkage editor to be loaded into memory and control passed to it. The linkage editor looks for its input on SYSLNK. The output from the linkage editor will be placed in the temporary portion of the core-image library unless the OPTION CATAL is specified.

This execute statement with no name in the operand field causes the program just placed in the temporary portion of the core-image library to be loaded into memory and executed.

These are the data cards that are to be processed by the program that was just compiled and link edited.

End of data cards.

End of job - all options reset to system standards.
1. Write the JCL to compile a COBOL program and produce only an object deck, a source listing, an error listing, and a cross-reference listing. Assume all options are set in the NO_ _ _ mode for all exercises in the course.

2. Write the JCL for a single job to compile two COBOL programs. The first should produce only an object deck and the second should produce only a source listing, an error listing, a cross-reference listing and a data division map.

3. Write the JCL to compile and execute a COBOL program. Produce a source listing, an error listing, a cross-reference listing and if the program abnormally terminates, a core dump should be provided.
The linkage editor combines object modules supplied in the input job stream, and/or newly compiled object modules on SYSLNK, and/or object modules from a relocatable library. It edits these modules into executable programs.

The linkage editor puts these executable programs into one of two places. If the option CATAL has been specified the program is cataloged as a member of a core image library. If not the program is placed in the temporary portion of the core-image library.

The compile-linkedit-execute example showed the linkage editor taking a just compiled program and linking it into the temporary portion of the core-image library.

If a program phase is to be cataloged into the core-image library a linkage editor control card is required. The card is a 'PHASE' card and the format is as follows.

```
PHASE name,origin
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ø</td>
<td>ø</td>
</tr>
</tbody>
</table>

Card column 1 must be blank.

PHASE The operation code

name 1 to 8 alphanemic characters used to give a name to the phase being cataloged. When the program phase is executed at a later date this is the name to be used in the execute card. (// EXEC name

,origin This operand allows the user to specify the load address to be used for this program phase. The possible options are;

1. symbol [(phase)][[relocation]
2. * [relocation]
3. S [relocation]
4. ROOT
5. + displacement
6. F + address

1. Used when building a program overlay structure. Says to link this phase at the same address as a previously linked phase with optional relocation shift.
2. Used when more than one phase is being linked together. Says put this one right behind the one just link edited. In a single phase operation this is the same as 'S'.

3. Says link this phase to load at the beginning of the partition we are working in.

4. Used in overlay and says that this particular phase is not to be overlayed but is to remain in memory while program is executing.

5. Specifies an absolute address.

6. Used to link edit a program to run in a foreground partition while the linkage editor is executing in background.

If object modules from the relocatable library or object decks external to the system are to be link edited, another control card comes into play.

```
 inclusion
```

```modulename```

modulename Symbol name of the module as used when cataloged in the relocatable library. If the operand is omitted the system assumes that the module is in the input job stream immediately behind the INCLUDE card.

The third linkage editor control card is the ENTRY card and is in the following format:

```
 ENTRY entrypoint
```

entrypoint Symbolic name of the address where control is to be passed when the program is loaded into memory.

Note: The ENTRY card is normally not needed. Job control will write an ENTRY statement when EXEC LNKEDT is read to ensure that an ENTRY statement will be present to halt linkage editing.
The last of the linkage editor control cards is ACTION card and is in the following format:

```
ACTION CLEAR, MAP, NOMAP, BG, F1, F2
```

CLEAR

Indicate that the unused portion of the core image library will be set to Binary zero before the beginning of the linkage editor function. CLEAR is very time-consuming and should not be used. If you are careful to assign initial values where needed in your program this isn't needed.

MAP

Indicates that PRINTER is available for diagnostic messages. In addition, a main storage map it output on PRINTER.

---

24/06/77

PHASE  XFR-AD  LOCORE  HICORE  DSK-AD  ESD  TYPE  LABEL  LOADED  REL-FR

PHASE*** 005000 005000 005AFO 30 13 3 CSECT  EXAMPLED 005000 005000
CSECT  IJCFZIIO 005950 005950
CSECT  IJDFAPIZ 005908 005908
* ENTRY  IJDFAZIZ 005908
CSECT  ILBDMN50 005AFO 005AFO

NOMAP

Diagnostics are listed on SYSLOG.

BG

Tells the linkage editor what

F1

Partition the program is being

F2

link edited to run in.

---

4-3
CSD.SFW.05
EXERCISES

1. You are handed a deck of cards and told that it is an object module received from Fort Knox DPI. Write the JCL necessary to execute the program.

2. Write the JCL needed to catalog the object module mentioned in exercise 1 into the core-image library. Call the program FXX512.

3. Write the JCL to execute the program cataloged in exercise 2.
ASSIGNMENT OF I/O DEVICES

In order for a program to run in a computer system some means must be available to link the program to the physical devices such as card readers, printers, and disk drives.

The programmer is given the latitude to utilize any symbolic device address he wishes, ranging from SYS000 through SYSmax. For example in a COBOL program you may choose to call the printer 'SYS005' and the card reader 'SYS006' and the disk unit you are using 'SYS010'. The preceding is accomplished through the use of the COBOL SELECT statement.

FILE-CONTROL
SELECT PRINT-FILE
ASSIGN TO SYS005-UR-1403-S.

Because this program could be required to run on many different hardware configurations the physical device addresses are likely to be different. The following diagram depicts a typical hardware configuration indicating the physical device addresses.

NOTE: The // ASSGN card discussed later in this section serves the function of linking the program logical assignments to the hardware configuration.
There exists in the DOS system a set of fixed symbolic names used to reference various I/O devices. These are:

- **SYSRDR** - Card reader, magnetic tape unit, or disk extent (specific area) used for JOB control input.
- **SYSIPT** - Card reader, magnetic tape or disk extent used as input for programs.
- **SYSPCH** - Card punch, magnetic tape or disk extent used for punched output.
- **SYSLST** - Printer, magnetic tape or disk extent used for punched output.
- **SYSLOG** - Printer-keyboard or console printer-keyboard used for operator messages and to log JCL message.
SYSLNK - Disk extent used as input to the linkage editor.

SYSRES - System residence area on a disk drive.

SYSCLCB - Disk extent used for a private core image library.

SYSRLB - Disk extent used for a private relocatable library.

SYSSLB - Disk extent used for a private source statement library.

SYSREC - Disk extent used to store error records collected by the system.

SYS000 - SYSmax - All other units on the system.

The first eleven of these symbolic names, called system logical units, are used by the system control programs and system service programs.

The remainder SYS000, SYS001, ......, SYSmax are known as programmer logical units. Some of these are defined and assigned to physical devices at system generation time such as the work areas required by the COBOL compiler.

The following are the physical device addresses for the ADMINCEN computer system and some of the Standard logical assignments set at system generation time:

<table>
<thead>
<tr>
<th>PHYSICAL ADDRESS</th>
<th>DEVICE</th>
<th>LOGICAL ASSIGNMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00C</td>
<td>CARD READER</td>
<td>SYSROR, SYSIPT, SYS020</td>
</tr>
<tr>
<td>00D</td>
<td>CARD PUNCH</td>
<td>SYSPCH, SYS021</td>
</tr>
<tr>
<td>00E</td>
<td>PRINTER</td>
<td>SYSLST, SYS022</td>
</tr>
<tr>
<td>01F</td>
<td>CONSOLE TYPEWRITER</td>
<td>SYSLNK, SYS000 thru SYS009</td>
</tr>
<tr>
<td>180-185</td>
<td>6 TAPE DRIVES</td>
<td>NONE</td>
</tr>
<tr>
<td>130</td>
<td>DISK DRIVE</td>
<td>SYSLNK, SYS000 thru SYS009</td>
</tr>
<tr>
<td>131</td>
<td>DISK DRIVE</td>
<td>SYSSLB, SYSCLCB</td>
</tr>
<tr>
<td>132</td>
<td>DISK DRIVE</td>
<td>NONE</td>
</tr>
</tbody>
</table>

Each of the disk assignments has associated with it information telling the system where exactly on the disk pack the specified file resides. For example, Disk drive 131 has eleven different areas on the pack assigned to SYSLNK and SYS000 thru SYS009, they all represent different areas some of which are used by the COBOL compiler for work areas.
The ASSGN statement assigns a logical I/O unit to a physical device. It remains in effect until the next change in assignment or until the end of job.

```
// ASSGN SYS005,X'00C'
```

SYSxxx The programmer logical unit used in your program.

address The physical device address in the form of X'cuu' which indicates the Channel and Unit number (in hexadecimal).

- CHANNEL (c)
  - '0' for multiplexor channel, 1-6 for selector channels 1 thru 6.
- UNIT (uu)
  - '00' - FE (0 to 254) in Hexadecimal

x'ss' used to tell the system the physical characteristics of a tape unit such as bytes per inch and parity used. Normally this information is provided at system generation time and is not needed unless changes have been made.

ALT Indicates an alternate magnetic tape unit that is used when the capacity of the original assignment is reached. The specifications are the same for primary and alternate tape drives.

```
// ASSGN SYS010,X'180'
// ASSGN SYS010,X'181', ALT
```

// ASSGN cards must be placed in the job stream prior to the // EXEC that utilizes them and should be placed after the previous // EXEC in the job stream. The reason for the latter is to group all of the associated JCL with the Job step. In the case of multistep jobs JCL changes are difficult if all of the 'ASSGN' cards are bunched together at the beginning of the first step.
AUTOMATIC DEVICE ASSIGNMENT METHOD (ADAM)

The DOS ASSGN that we have just discussed does have some limitations. One of the limitations is that the JCL, which is written ahead of time, is hardware system dependent and would have to be modified card-by-card if the same set of JCL were to be used on a different hardware configuration. This does not present a great problem for programs that are written and run on the same machine. But, it does present a problem for Army Standard systems that are written in a central location along with the associated JCL and sent out to the world to be run on many different machines. A second limitation is the ability to get around an I/O device malfunction. It must be done in the same manner as above, a card-by-card change in the JCL ASSGN's.

To overcome these limitations ADAM was written to provide the ability to make DOS JCL hardware configuration independent and return system resources to the DOS console operator for management.

EXAMPLE: // ASSGN SYS010,X 'TP0'

TP0 Positions 1 and 2 indicate the particular device type;
CR card reader
CP card punch
PR printer
DK disk drive
TP tape drive
TD tape drive w/dual density feature
T7 tape drive seven track

The 3rd position indicates the positional relationship of this device to any other device of the same classification. For example; TP0,TP1,TP2,TP3,.... or CR0,CR1,PR0,PR1,....

There is a SYMBOLIC UNIT TABLE in memory that relates these symbolic assignments to the actual device addresses. The table logically appears as follows;

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>BG</th>
<th>F2</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR0</td>
<td>00C</td>
<td>00C</td>
<td>00C</td>
</tr>
<tr>
<td>CP0</td>
<td>00D</td>
<td>00D</td>
<td>00D</td>
</tr>
<tr>
<td>PR0</td>
<td>00E</td>
<td>00E</td>
<td>00E</td>
</tr>
<tr>
<td>TP0</td>
<td>180</td>
<td>182</td>
<td>184</td>
</tr>
<tr>
<td>TP1</td>
<td>181</td>
<td>183</td>
<td>185</td>
</tr>
<tr>
<td>TP2</td>
<td>182</td>
<td>184</td>
<td>180</td>
</tr>
<tr>
<td>TP3</td>
<td>183</td>
<td>185</td>
<td>181</td>
</tr>
<tr>
<td>TP4</td>
<td>184</td>
<td>180</td>
<td>182</td>
</tr>
<tr>
<td>TP5</td>
<td>185</td>
<td>181</td>
<td>183</td>
</tr>
<tr>
<td>DK1</td>
<td>130</td>
<td>134</td>
<td>136</td>
</tr>
</tbody>
</table>

In the example above we saw that the program called for 'TP0'. As you can see from the table if the program were run in the background (BG) partition it would be assigned to tape drive '180'. If it were run in foreground 2 (F2) it would be assigned to tape drive '182'. On some other hardware configuration it may be an address that does not even appear in this table.
If device 180 were to become inoperative for some reason, the operator would change the assignment in the symbolic unit table with the following three commands entered thru the console device:

```
SYMCHG TP0,BG=185
SYMCHG TP4,F2=183
SYMCHG TP2,F1=181
```

This is certainly easier than hunting up all of the ASSGN SYSxxx,X'180' cards and repunching them.

The Symbolic Unit Table takes about 30 seconds to build at IPL time.
1. Write the JCL to compile and execute a COBOL program which uses the following SELECT statements

SELECT CARD-FILE ASSIGN SYS008-UR-2540R-S.
SELECT PRINT-FILE ASSIGN SYS009-UR-1403-S.

2. Write the JCL to execute an object deck for a program which reads cards and produces a magnetic tape file and a printed report. The programmers logical units used by the program are SYS005 for the card reader, SYS008 for the printer, and SYS010 for the tape file.

3. Write the JCL for two COBOL programs which are to be compiled and executed as separate job steps within the same job. The first program while executing reads a card file and punches an output card file. The second program reads another set of data cards and produces a printed report. The first program uses SYS005 for the card reader and SYS007 for the card punch. The second program uses SYS006 for the card reader and SYS005 for the printer.
DATA FILES

The diagram below is a portion of a Systems Flow Chart. It depicts a single program and its associated inputs and outputs.

We have already seen how the // ASSGN ... cards are used to tell the operating system what devices are to be assigned to this program in order to have all of these input and output files on line. No further discussion is necessary for the CARD and PRINT files. However, because there are normally hundreds or even thousands of reels of tape and hundreds of disk packs in an organization, it is necessary to have some way of uniquely identifying each disk pack and each tape reel. The problem is compounded by the fact that there may be more than one file on each volume.

TAPE FILES

(IBM TAPES LABELS FOR DOS; ORDER NO. GC24-5070)

Tape files may appear as one or more files per volume or more than one
volume per file. In any case the system must have some means of identifying the volume that it is reading and some means of identifying the file on that volume(s). This is accomplished through the use of STANDARD TAPE LABELS. Each volume has a standard volume label and a standard end-of-volume label. Each file has a standard header label and a standard end-of-file label. (See Page 6-2.1 Volume Layouts)

**VOLUME-LABEL**

The VOL1 label is written by a utility program, Initialize Tape. It is generally written once, when the reel of tape is first received in an installation. At that time, a permanent volume serial number is assigned to the reel and written on it as part of the volume label. This provides a permanent identification of the reel, as long as it is used for files, with standard labels. (See Page 6-2.2 Standard Volume Label)

**FILE-HEADER-LABEL**

Each tape file has a header label as the first physical record. It consists of 80 bytes of information as provided by the user and the IOCS routines to uniquely identify the file. (See Page 6-2.3 Standard File Header Label)

**END-OF-FILE OR END-OF-VOLUME LABEL**

Either one or the other of these will appear at the end of each volume depending on the file-volume relationship. The data in these labels are the same as that in the File-Header-Label with two exceptions. Bytes 1 thru 4 will contain either EOF1 or EOV1 and bytes 55 thru 60 contain a block count which indicates the number of physical records written on it's particular volume.
SINGLE-FILE-SINGLE-VOLUME

FIRST VOLUME

MIDDLE VOLUME(S)

LAST VOLUME

SINGLE-FILE-MULTIVOLUME

MULTIPLE-SINGLE-VOLUME

NOTE: TM is tape mark which is a special character that the tape drive writes on tapes to indicate check points.
<table>
<thead>
<tr>
<th>VOL1</th>
<th>serial number</th>
<th>not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BYTES</th>
<th>ENTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-4</td>
<td>VOL1</td>
</tr>
<tr>
<td>5-10</td>
<td>volume serial number written when tape is initialized by Utility Program.</td>
</tr>
<tr>
<td>11-80</td>
<td>not used for DOS</td>
</tr>
<tr>
<td>Bytes</td>
<td>Entry</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>1-4</td>
<td>HDR1</td>
</tr>
<tr>
<td>5-21</td>
<td>File Identifier</td>
</tr>
<tr>
<td>22-27</td>
<td>File Serial Number</td>
</tr>
<tr>
<td>28-31</td>
<td>Volume Sequence Number</td>
</tr>
<tr>
<td>32-35</td>
<td>File Sequence Number</td>
</tr>
<tr>
<td>36-39</td>
<td>Generation Number</td>
</tr>
<tr>
<td>40-41</td>
<td>Version Number of Generation</td>
</tr>
<tr>
<td>42-47</td>
<td>Creation Date</td>
</tr>
<tr>
<td>48-53</td>
<td>Expiration Date</td>
</tr>
</tbody>
</table>
The tape label card is used to provide file label information for tape label checking and writing.

**FORMAT**

```plaintext
// TLBL
filename,['file-id'],[date],[file-serial-number],
[volume-sequence-number],[file-sequence-number],
[generation-number],[version-number]
```

**filename**

This operand serves to link the TLBL card to the COBOL program file description (FD). It consists of 1 to 7 alphameric characters, the first of which must be an alphabetic.

**EXAMPLE 1:**

```plaintext
SELECT TAPE-FILE1
    ASSIGN TO SYS011-UT-2400-S-TAPEFLE.
// ASSGN SYS011,X'183'
// TLBL TAPEFLE,'TAPE FILE ONE'
```

**EXAMPLE 2:**

```plaintext
SELECT TAPE-FILE2
    ASSIGN TO SYS012-UT-2400-S.
// ASSGN SYS012,X'184'
// TLBL SYS012,'TAPE FILE TWO'
```

**'file-ID'**

This optional entry consists of 1 to 17 alphameric characters, contained within apostrophes. It appears in the TLBL and the file label and serves to uniquely identify the file.

**date**

This optional entry enables the user to specify how long the file is to be retained and can be specified in one of three ways:
1. Omit operand in which case the file will not be retained.

2. Enter an absolute expiration date in the form of yy/ddd.
   84/005 file will expire on 5 January 1984
   82/042 file will expire on 11 February 1982

3. Enter the number of days that the file is to be retained in the form of 0 to 9999.
   9 file will expire 9 days after creation.
   27 file will expire 27 days after creation.
   9999 is used to specify a permanent file.

file serial
number

This optional entry allows the user to specify the volume serial number of the first or only reel of tape in file.

volume
sequence
number

If desired on input files you may wish to use this optional operand to access some volume of a multivolume other than the first. Enter 1 to 4 numeric characters. (This operand is usually omitted)

file
sequence
number

This optional entry allows the user to access a specific file within a multifile volume. Enter 1 to 4 numeric characters.
Generation number
Optional operand for the user to specify a generation number.
Enter 1 to 4 numeric characters.

Version number
Optional operand to sub-identify a particular generation.
Enter 1 or 2 numeric characters.

// LBLTYP TAPE or // LBLTYP NSD(02)

The LBLTYP statement (reserve storage for label processing) defines the amount of main storage to be reserved at linkage-edit time.

FORMAT

// LBLTYP TAPE
NSD(nn)

TAPE Used only if tape files requiring label information are to be processed, and no nonsequential DASD files are to be processed.

NSD(nn) Used if any nonsequential DASD files are to be processed regardless of other type files to be used. nn specifies the largest number of extents to be used in a single file.

The amount of storage that must be reserved for label information is:
1. For standard tape labels 80 bytes.
2. For sequential disk labels NONE.
3. For ISAM or DA disk file labels 84 bytes plus 20 bytes per extent.
1. Write the JCL to execute a program that is cataloged in the core image library under the phase-name 'WS342' and contains the following select statements:

SELECT CARD-FILE
ASSIGN TO SYS008-UR-2540R-S.

SELECT TAPE-FILE
ASSIGN TO SYS011-UT-2400-S.

TAPE-FILE is an output file. Utilize 'ERROR FILE TAPE' as a unique file ID and put the file on tape SN 772206. Retain the file for one year.

2. Write the ASSGN's and TLBL's for a program containing the following select statements:

SELECT TAPE-INPUT-FILE
ASSIGN TO SYS011-UT-2400-S.

SELECT TAPE-OUTPUT-FILE
ASSIGN TO SYS012-UT-2400-S-NEWMST.
TAPE-INPUT-FILE has no unique file ID but is on tape SN 750349.

TAPE-OUTPUT-FILE is to go on tape SN 781433. Assign your own file ID. The file is to be retained indefinitely.
STANDARD MASS STORAGE DEVICE LABELS -- (VTOL)

Expiration Date
Spare

Creation Date

File Name
File Serial Number
System Code

Format Identifier
Volume Sequence Number
Count of Extents
Bytes used in Last block of Directory

Record Length
Option Code
Key Location
Space Remaining

FIRST EXTENT
ADDITIONAL EXTENT
ADDITIONAL EXTENT
The disk label card is used to provide part of the label information for disk label writing and checking.

**FORMAT**

// DLBL filename,['file id'],[date], [code]

**filename**  This operand serves to link to DLBL card to the COBOL program file description (FD). It consists of 1 to 7 alphanumerical characters, the first of which must be alpha.

Example: SELECT OM-DISK-MASTER-FIELD ASSIGN TO SYS007-UT-2314-S-DISK5.

// ASSGN SYS007,X'132'

// DLBL DISK5,'Mast........

**file id**  This operand links the DLBL card to the data set on the disk pack. It is 1 to 44 characters in length, the first of which must be alpha. The file id must be unique on any given disk pack.

**date**  The date is used when creating an output file and can have one of two formats;

- yy/ddd using year and julian date to specify expiration date.
- dddd to specify the number of days the file is to be retained.

**code**  This operand tells the system what kind of file this is;

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>Sequential disk file</td>
</tr>
<tr>
<td>DA</td>
<td>Direct access file</td>
</tr>
<tr>
<td>ISC</td>
<td>Indexed sequential (used only when creating file)</td>
</tr>
<tr>
<td>ISE</td>
<td>Indexed sequential (used for all other activities)</td>
</tr>
</tbody>
</table>

// EXTENT SYS007,123321,1,0,123,1520

The disk extent(area) card is used to tell the system such things as what disk pack the file is on, where on that pack does the file begin and how many tracks does the file occupy.

**FORMAT**

// EXTENT [symbolic unit],[serial number],[type],[sequence number],[relative track],[number of tracks]
symbolic unit  This operand must match the symbolic unit in its associated assign card.

serial number  Volume serial number of the disk pack for which this extent is effective.

type  This operand specifies the type of extent this is;

<table>
<thead>
<tr>
<th>TYPE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prime data area for any of the 3 file types.</td>
</tr>
<tr>
<td>2</td>
<td>Independent overflow for ISAM files.</td>
</tr>
<tr>
<td>4</td>
<td>Index area for ISAM.</td>
</tr>
</tbody>
</table>

sequence number  For sequential disk files the first extent is sequence number 0, the second is 1, the third is 2 and so on.

For ISAM files the following rules apply;

0    Master index only
1    Cylinder index only
2    first prime data extent
3    second prime data extent
etc... 4, 5, 6... N
N+1    Independent overflow

relative track  This operand indicates the starting track of the file relative to the first track on the disk pack.

The user will normally know the cylinder and track number. If so, you will have to convert it to relative track using the following formula;

relative track (RT) = 20 \times \text{(cylinder number)} + \text{(track number)}

EXAMPLE: Your sequential disk file begins on cylinder number 15 track number 5. What is the relative track?

RT = 20 \times 15 + 5
RT = 305

NOTE: For ISAM files prime data extents, the relative track must be a multiple of 20.

number of tracks  This operand is the total number of tracks in this extent.

EXAMPLE: Your sequential disk file begins on cylinder 15 track 5 and ends on cylinder 21 track 15. What is the number of tracks?

RT(start) = 20 \times 15 + 5 = 305, RT(end) = 20 \times 21 + 15 = 435
number of tracks = RT(end) - RT(start) + 1 = 435 - 305 + 1 = 131
Requirement: You are to write the job control language to execute the program called MAKFILE, which is currently cataloged in the core image library.

The sequential disk file is to be called SEQUENTIAL FILE 53 and is to be placed on disk pack number 555663 which is to be mounted on disk drive 132. The file begins on cylinder 105 track 3 and ends on cylinder 112 track 7. The file is to be retained for 20 days. The COBOL program selects are as follows:

```cobol
FILE-CONTROL.
SELECT IT-CARD-FILE
   ASSIGN TO SYS004-UR-2540R-S.
SELECT OR-MAKFILE-REPORT
   ASSIGN TO SYS005-UR-1403-S.
SELECT OM-DISK-FILE
   ASSIGN TO SYS010-DA-2314-S-SEQFILE.
```
// JOB EXSEQ
// OPTION LOG,DUMP
// ASSGN SYS004,X'00C'
// ASSGN SYS005,X'00E'
// ASSGN SYS010,X'132'
// DLBL SEQFLE,'SEQUENTIAL FILE 53',20,SD
// EXTENT SYS010,555663,1,0,2103,145
// EXEC MAKFILE

INPUT DATA
DECK

/*

/&

RT(start) = 20 X 105 + 3 = 2103
RT(end) = 20 X 112 + 7 = 2247

Number of tracks = 2247 - 2103 + 1 = 145
EXERCISES

1. Write the JCL for a program called 'MAKSEQ' that reads cards and creates a sequential disk file. The file is to be stored on disk pack 122013 which will be mounted on drive 134. It will be stored from cylinder 10 track 0 through cylinder 119 track 19. The file should expire 150 days after it is created. File-ID is CUSTOMER MASTER FILE.

   SELECT CUST-MASTER-FILE
   ASSIGN TO SYS012-UT-2314-S.
   SELECT CARD-INPUT-FILE
   ASSIGN TO SYS005-UR-2540R-S.

2. Write the JCL for a card to disk program. The COBOL program is to be compiled and executed. The file is to be stored on disk pack 506012 which will be mounted on drive 132. It will be stored from cylinder 24 track 0 through cylinder 49 track 19. The file-ID is PRICE-MASTER-FILE. The file is to be retained indefinitely.

   SELECT CARD-MASTER-FILE
   ASSIGN TO SYS026-UR-2540R-S.
   SELECT PRICE-MASTER-FILE
   ASSIGN TO SYS024-UT-2314-S-PRIMAST.
I. REQUIREMENT. Write the DOS JCL necessary to execute your COBOL PE number 2 or 3. You are expected to get the same compiler outputs as before as well as a clean execution of your program.

II. COBOL program changes. There are some differences between DOS and OS COBOL. In order to get a clean compile you will have to do the following:

1. Change the disk file assign to read
   ASSIGN TO SYS006-UT-2314-S.

2. Change the printer file assign to read
   ASSIGN TO SYS005-UR-1403-S.

3. Change the printer FD label clause to read
   LABEL RECORDS ARE OMITTED

4. Remove the printer FD BLOCK CONTAINS clause.

III. METHODS.

1. Replace the // JOB name card with the following card filled out the same way as your normal computer run card;

```
LAST NAME, FIRST  ROOM = nnnn  (please punch)

U.S. ARMY INSTITUTE OF ADMINISTRATION
COMPUTER SCIENCE DEPARTMENT

FOR USE ONLY WITH EXERCISE JO1

DOS
FOR INSTRUCTIONAL PURPOSES ONLY
```

2. Be sure all of your JCL is listed on the printer.
3. Do not use the DECK or CATAL options.
4. The disk pack is to be mounted on disk drive 'DK3'.
5. The output is to be printed on printer 'PR0'.
6. The serial number of the disk pack is DOSWK3.
7. The file ID is 'SOFTWARE DATA FILE 14 * RESIDENT'.

7-7  CSD.SFW.OS
INDEXED SEQUENTIAL ACCESS METHOD (ISAM) FILES

An ISAM file is basically a sequential file with a series of indexes that allow the processing of single records located anywhere within the file. An ISAM file can also be processed sequentially if desired. There are several different types of areas associated with an ISAM file and these are explained below (See diagram on the next page as you read).

PRIME AREA is where the records are located and may occupy one cylinder or many disk packs. The prime area is defined by the programmer thru JCL. The records are located in this area in sequential order (logically) by record key in ascending order. A KEY is some unique field within the logical record. In a personnel file perhaps we would use Social Security Number.

INDEX AREA is the place on the disk pack where ISAM builds and maintains a series of indexes used to access or add records to the file when processing in a random mode.

TRACK INDEX occupies the first track of each cylinder of the prime area. The track index is defined by ISAM and as with all other indexes is loaded by ISAM. Its' function is to tell ISAM on which track of this cylinder the record is located or if being added, on which track it is to be added to.

CYLINDER INDEX is defined by the programmer thru JCL. Its' function is to tell ISAM on which cylinder the record should be located.

MASTER INDEX is defined by the programmer using JCL. Its' function is to tell ISAM which track of the cylinder index to look for this particular record. The master index is the only optional index in the ISAM file. It is normally defined if the cylinder index is greater than 4 tracks.

OVERFLOW AREA is used when records are added to an already established file.

CYLINDER OVERFLOW. A certain number of whole tracks are reserved in each cylinder for overflow records from the prime area. The programmer may specify the number of tracks to be reserved by means of the APPLY CYL-OVERFLOW clause. If zero is specified no cylinder overflow will be reserved. If the clause is omitted, ISAM defaults to 20% of each cylinder. This area is not defined by JCL.

INDEPENDENT OVERFLOW AREA. Overflow records from anywhere in the prime area are placed in a certain number of cylinders reserved solely for this purpose. The size and location of this area is defined by the programmer thru JCL.
Elements of an Indexed Sequential (ISAM) File

Pointer to a track of the cylinder index.

Master Index

Cylinder Index

Pointer to a cylinder in the prime area.

Pointer to a track in the data area.

Track Index

Data Area

Cylinder Overflow Area

Overflows from the cylinder overflow goes to the independent overflow.

Overflows (additions) from the data area will be placed in the cylinder overflow.

Index Area

Prime Data Area

Independent Overflow Area
Requirement: You are to write the job control language needed to execute the program called CREATE, which is currently cataloged in the core image library. The systems flow chart below will assist you by showing you the relationships of all files.

The transaction tape file has a file Id of 'TRANS TAPE 156' and the file serial number is 556783. It is to be mounted on drive 182.

The ISAM file occupies 3 disk packs as follows:
- The MASTER INDEX is on disk pack number 122375 and begins on cylinder 15 track 3 and ends on cylinder 15 track 5. Mount pack on drive 131.
- The CYLINDER INDEX, as required, begins on the next track after the master index and takes 12 tracks.
- The PRIME AREA is on disk pack number 155732 and occupies cylinders 5 thru 167. Mount that pack on drive 132.
- The INDEPENDENT OVERFLOW AREA is on disk pack number 123723 and occupies all of cylinder 12. Mount that pack on drive 133.
Retain the file for 180 days and call it 'ISAM MASTER FILE 156'.

The COBOL program selects are as follows:

FILE-CONTROL.
SELECT IT-TRANS-TAPE-FILE
   ASSIGN TO SYS010-UT-2400-S-SQTAPE.
SELECT OM-MASTER-FILE
   ASSIGN TO SYS015-DA-2314-I-ISMAST
   ACCESS IS SEQUENTIAL
   RECORD KEY IS OM01-SSAN.
SELECT OR-MASTER-CREATE-RPT
   ASSIGN TO SYS005-UR-1403-S.
I-O-CONTROL.
APPLY CYL-OVERFLOW OF 2 TRACKS ON OM-MASTER-FILE 7-10
SOLUTION:

// JOB ISAMEX
// OPTION LOG,DUMP

// ASSGN SYS005,X'OE'
// ASSGN SYS010,X'182'
// TLBL SQTape,'TRANS TAPE 156',556783
// ASSGN SYS015,X'131'
// ASSGN SYS020,X'132'
// ASSGN SYS025,X'133'
// DLBL ISMAST,'ISAM MASTER FILE 156',180,ISC
// EXTENT SYS015,122375,4,303,3
// EXTENT SYS015,122375,4,306,12
// EXTENT SYS020,155732,1,3260
// EXTENT SYS025,123723,2,240,20
// EXEC CREATE

/\
1. Write the DLBL and EXTENT cards to create an ISAM file. The prime data area will be the area from cylinder 20 track 0 through cylinder 119 track 19. The master index will use cylinder 140 track 0 through cylinder 140 track 3. The cylinder index will use cylinder 140 track 4 through cylinder 140 track 15. The file is to be stored on pack 223556 and is to be saved for 500 days. Use the following select statement.

   SELECT INVENTORY-MASTER-FILE
      ASSIGN TO SYS009-DA-2314-I-MSTFLE.

2. Write the ASSGN, DLBL, AND EXTENT cards to create an ISAM file on disk pack 443556 drive 135. The file should not expire. The prime data area will be cylinder 29 track 0 through cylinder 35 track 19 and the cylinder index will be from cylinder 28 track 0 through cylinder 28 track 2. Use the following select statement.

   SELECT CUSTOMER-MASTER
      ASSIGN TO SYS005-DA-2314-I-CUSMSTR.
EXERCISES

3. Write the JCL to compile and execute a COBOL program which is to randomly retrieve and update the master file created in problem #2. The file-ID for the transaction file is TRANSACTION FILE and the serial number is 100005. No other information is needed for checking this label. Disk drive 134 and Tape drive 183 are to be used. Use the following COBOL select statements.

SELECT MASTER-FILE
   ASSIGN TO SYS021-DA-2314-I-CUSMSTR.
SELECT TRANS-FILE
   ASSIGN TO SYS022-UT-2400-S-TRNSFLE.
As we have seen the DOS system consists of three libraries; the core image library, the relocatable library, and the source statement library. We have also seen some of the functions performed on the libraries.

1. The linkage editor places the link edited phases into the core image library either in the temporary portion for immediate execution or catalog them as permanent members for execution at any time.

2. The system loader finds executable phases in the core image library and loads them into memory for execution.

3. Link editor will load modules into SYSLNK from the relocatable library to be link edited.

4. The linkage editor will extract relocatable IOCS modules from the relocatable library and attach them (link edit) to COBOL Programs.

5. The COBOL compiler and the assembler have the capability of extracting source code from the Source statement library and combining it with other source code.
FUNCTIONS OF LIBRARIAN

1. Maintenance functions add, delete, or rename components of the three libraries, condense or remove spaces that may exist in the directories or libraries themselves. Alter the amount of space allocated for each library. The program phase that performs this maintenance function is called MAINT and it applies to all three libraries.

2. Service functions will print and/or punch library members and directories. There all four program phases that perform these services and they are:

   DSERV - library directories (all)
   CSERV - core image library
   RSERV - relocatable library
   SSERV - Source statement library

3. Copy functions will either completely or selectively; copy the disk pack on which the system resides, create private libraries, or merge libraries. The program phase that copies libraries is called CORGZ.
1. DSERV: the purpose of DSERV is to list, on the printer, the contents of the directories of the three DOS libraries in a readable format.

Format of control card:

```
% DSPLY directory [ ,directory]

directory: can be one of the following:

TD
CD
RD
SD
ALL
```

EXAMPLE: The following example would be used to cause the system to print out the directory for the core image library.

```
// JOB CORING
// EXEC DSERV
% DSPLY CD
/*
/&
```
<table>
<thead>
<tr>
<th>PHASE NAME</th>
<th>DISK ADDR</th>
<th>NO.</th>
<th>SIZE</th>
<th>LOAD ADDR</th>
<th>ENTRY ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLCRSEQE</td>
<td>034 18 03</td>
<td>001</td>
<td>0288</td>
<td>005000</td>
<td>005000</td>
</tr>
<tr>
<td>FLCRKNEE</td>
<td>034 18 04</td>
<td>001</td>
<td>0704</td>
<td>005000</td>
<td>005000</td>
</tr>
<tr>
<td>H43ATP</td>
<td>034 19 01</td>
<td>007</td>
<td>0720</td>
<td>000000</td>
<td>006080</td>
</tr>
<tr>
<td>P23ATPLE</td>
<td>035 00 04</td>
<td>001</td>
<td>1601</td>
<td>000000</td>
<td>0005F0</td>
</tr>
<tr>
<td>P23SPLLE</td>
<td>035 01 01</td>
<td>007</td>
<td>0209</td>
<td>000648</td>
<td>002148</td>
</tr>
<tr>
<td>P23SPULF</td>
<td>035 02 04</td>
<td>020</td>
<td>0205</td>
<td>007E90</td>
<td>00A608</td>
</tr>
<tr>
<td>PW44OK1</td>
<td>035 07 04</td>
<td>018</td>
<td>0409</td>
<td>000000</td>
<td>0068AF</td>
</tr>
<tr>
<td>P23ATP</td>
<td>035 12 02</td>
<td>001</td>
<td>1378</td>
<td>0360CC</td>
<td>0360CC</td>
</tr>
<tr>
<td>SUGSPOOL</td>
<td>035 12 03</td>
<td>006</td>
<td>1659</td>
<td>036288</td>
<td>038128</td>
</tr>
<tr>
<td>SUGSWPOER</td>
<td>035 14 01</td>
<td>015</td>
<td>0894</td>
<td>0380A8</td>
<td>03D7FA</td>
</tr>
<tr>
<td>P23ATP30</td>
<td>035 17 04</td>
<td>001</td>
<td>1364</td>
<td>0360CC</td>
<td>0360CC</td>
</tr>
<tr>
<td>P23SSP30</td>
<td>035 18 01</td>
<td>006</td>
<td>1659</td>
<td>03618A</td>
<td>039118</td>
</tr>
<tr>
<td>P23SPW30</td>
<td>035 19 03</td>
<td>015</td>
<td>0852</td>
<td>038909</td>
<td>03A909</td>
</tr>
<tr>
<td>P23ATP40</td>
<td>036 03 07</td>
<td>001</td>
<td>1641</td>
<td>000000</td>
<td>0005F8</td>
</tr>
<tr>
<td>P23SSP40</td>
<td>036 03 03</td>
<td>007</td>
<td>0209</td>
<td>006790</td>
<td>002170</td>
</tr>
<tr>
<td>P23SPW40</td>
<td>036 05 02</td>
<td>018</td>
<td>0401</td>
<td>002E90</td>
<td>009 AAP</td>
</tr>
<tr>
<td>P24ATS</td>
<td>036 09 04</td>
<td>020</td>
<td>0912</td>
<td>000000</td>
<td>00653A</td>
</tr>
<tr>
<td>$262ATP5</td>
<td>036 14 04</td>
<td>013</td>
<td>0218</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>$262ATP4</td>
<td>036 18 01</td>
<td>013</td>
<td>0218</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>$262AFAJ4</td>
<td>037 01 02</td>
<td>013</td>
<td>0218</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>$262AFHA6</td>
<td>037 04 03</td>
<td>013</td>
<td>0218</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>$262AEE2</td>
<td>037 07 04</td>
<td>019</td>
<td>0298</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>$262AJR</td>
<td>037 15 04</td>
<td>013</td>
<td>0218</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>$262AMTP</td>
<td>037 19 01</td>
<td>019</td>
<td>0298</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>$262AP4</td>
<td>038 03 04</td>
<td>013</td>
<td>0218</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>$262AT4I4</td>
<td>038 07 01</td>
<td>013</td>
<td>0218</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>FC880C</td>
<td>038 10 02</td>
<td>006</td>
<td>1659</td>
<td>000000</td>
<td>001800</td>
</tr>
<tr>
<td>FC880L10</td>
<td>038 11 04</td>
<td>017</td>
<td>0650</td>
<td>0007800</td>
<td>008470</td>
</tr>
<tr>
<td>FC880L12</td>
<td>038 16 01</td>
<td>017</td>
<td>1428</td>
<td>00A3600</td>
<td>00436A</td>
</tr>
<tr>
<td>FC880L14</td>
<td>039 00 07</td>
<td>010</td>
<td>1216</td>
<td>00A3600</td>
<td>00436A</td>
</tr>
<tr>
<td>FC880L11</td>
<td>039 05 01</td>
<td>016</td>
<td>1400</td>
<td>00A3600</td>
<td>00436A</td>
</tr>
<tr>
<td>FC880L20</td>
<td>039 09 01</td>
<td>013</td>
<td>1264</td>
<td>00A15A8</td>
<td>00A168</td>
</tr>
<tr>
<td>FC880L22</td>
<td>039 12 07</td>
<td>015</td>
<td>0384</td>
<td>00A15A8</td>
<td>00A168</td>
</tr>
<tr>
<td>FC880L21</td>
<td>039 16 01</td>
<td>016</td>
<td>0404</td>
<td>00A15A8</td>
<td>00090C</td>
</tr>
<tr>
<td>FC880L30</td>
<td>040 01 01</td>
<td>007</td>
<td>1540</td>
<td>00A15A8</td>
<td>000C7A8</td>
</tr>
<tr>
<td>FC880L40</td>
<td>040 01 04</td>
<td>023</td>
<td>1583</td>
<td>0090A0</td>
<td>012474</td>
</tr>
<tr>
<td>FC880L50</td>
<td>040 07 03</td>
<td>027</td>
<td>0048</td>
<td>000000</td>
<td>0011470</td>
</tr>
<tr>
<td>FC880L51</td>
<td>040 13 01</td>
<td>021</td>
<td>0250</td>
<td>0090A0</td>
<td>00113A8</td>
</tr>
<tr>
<td>FC880L60</td>
<td>040 19 02</td>
<td>016</td>
<td>1426</td>
<td>0090A0</td>
<td>0090A0</td>
</tr>
<tr>
<td>FC880L70</td>
<td>041 02 02</td>
<td>032</td>
<td>0579</td>
<td>0090A0</td>
<td>0090A4</td>
</tr>
<tr>
<td>R262AP4</td>
<td>041 09 02</td>
<td>021</td>
<td>0219</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>R262AT4I4</td>
<td>041 12 04</td>
<td>004</td>
<td>0404</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>R262AT4I4</td>
<td>041 13 04</td>
<td>004</td>
<td>0944</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>R262AT4I4</td>
<td>041 14 04</td>
<td>004</td>
<td>0453</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>R262AT4I4</td>
<td>041 15 04</td>
<td>004</td>
<td>0744</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>R262AT4I4</td>
<td>041 16 04</td>
<td>004</td>
<td>0494</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>

See page 8-7 for more information.
### System Directory

- **Date:** 24/01/78

<table>
<thead>
<tr>
<th>Directory Starting Address</th>
<th>Core-Image</th>
<th>Relocatable</th>
<th>Source-Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 10 01</td>
<td>01 10 01</td>
<td>01 10 01</td>
<td>01 10 01</td>
</tr>
<tr>
<td>00 13 11 04</td>
<td>04 13 11 04</td>
<td>04 13 11 04</td>
<td>04 13 11 04</td>
</tr>
<tr>
<td>00 15 15 17</td>
<td>17 15 15 17</td>
<td>17 15 15 17</td>
<td>17 15 15 17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Library Starting Address</th>
<th>Core-Image</th>
<th>Relocatable</th>
<th>Source-Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 16 01</td>
<td>01 16 01</td>
<td>01 16 01</td>
<td>01 16 01</td>
</tr>
<tr>
<td>42 11 01</td>
<td>02 11 01</td>
<td>02 11 01</td>
<td>02 11 01</td>
</tr>
<tr>
<td>45 19 04</td>
<td>04 19 04</td>
<td>04 19 04</td>
<td>04 19 04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Information</th>
<th>Core-Image</th>
<th>Relocatable</th>
<th>Source-Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory Entries Active</td>
<td>975</td>
<td>609</td>
<td>137</td>
</tr>
<tr>
<td>Library Blocks Allocated</td>
<td>3616</td>
<td>4720</td>
<td>20466</td>
</tr>
<tr>
<td>Library Blocks Active</td>
<td>3288</td>
<td>3357</td>
<td>14373</td>
</tr>
<tr>
<td>Library Blocks Deleted</td>
<td>57</td>
<td>180</td>
<td>545</td>
</tr>
<tr>
<td>Library Blocks Available</td>
<td>276</td>
<td>1133</td>
<td>5543</td>
</tr>
<tr>
<td>Automatic Condense Limit</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>Library Allocated Cylinders</td>
<td>45</td>
<td>15</td>
<td>38</td>
</tr>
<tr>
<td>Directory Allocated Tracks</td>
<td>06</td>
<td>05</td>
<td>02</td>
</tr>
</tbody>
</table>

---

**CSD-SFM-OS**
CSERV: The purpose of CSERV is to print out on the printer and/or punch on the card punch the members of the core image library.

Format of control card:

\[ \text{operation \ operand [ , operand 2]} \]

Operation: May be one of the following:

- DSPLY
- PUNCH
- DSPCH

Operand: May be one of the following:

- phasename
- progl.ALL
- ALL

EXAMPLE: Print out all of the phases of FCOBOL

// JOB PRTFCOB
// EXEC CSERV
\[ 0 \text{ DSPLY FCOB.ALL} \]
/*
*/
/*
00A150
00A180
00A1P0
00A1F0
00A210
00A240
00A270
00A2D0
00A2H0
00A300
00A330
00A360
00A390
00A3C0
00A3F0
00A420
00A450
00A480
00A4B0
00A4E0
00A510
00A540
00A570
00A5A0
00A5D0
00A600
00A630
00A660
00A690
00A6C0
00A6F0
00A720
00A750
00A780
00A7B0
00A7E0
00A810
00A840
00A870
00A8A0
00A8D0
00A900
00A930
00A960
00A990
00A9C0
00A9F0
00A020
00A050
00A080
00A0B0
00A0E0
00A110
00A140
00A170
00A1A0
00A1D0
00A1F0
00A220
00A250
00A280
00A2B0
00A2E0
00A310
00A340
00A370
00A3A0
00A3D0
00A400
00A430
00A460
00A490
00A4C0
00A500
00A530
00A560
00A590
00A5C0
00A5F0
00A620
00A650
00A680
00A6B0
00A6E0
00A710
00A740
00A770
00A7A0
00A7D0
00A800
00A830
00A860
00A890
00A8C0
00A8F0
00A920
00A950
00A980
00A9B0
00A9E0
00A9H0
00A940
00A970
00A9A0
RSERV: Performs the print and punch functions for the relocatable library.

Format of control card

$ operation $ operand1 [ , operand2]

Operation:

DSPLY
PUNCH
DSPCH

Operand:

Modulename
progl.ALL
ALL

EXAMPLE: Punch out the printer IOCS module called IJDFAPIZ

// JOB PRTPTR
// EXEC RSERV
$ PUNCH IJDFAPIZ
/*
/&
WARP
vim
FCD
TXT

1.,)
000100
032
136
0001
0001
00001 BLOCKS
SYSTEm
RELOCATABLE LIBRARY

F0240AC7 90CE0EDC 58E0101F 06E092CD 1017F000 18C180DC 43C01017
43DE0F7F 19CF4780 F05046F0 F0409A32 47NEF107 43C01016 19CF4770
10164780 FC76920F 10280400 91801732 4710F076 0A074200 10284200
91801002 4710F08A

TXT
00088 136 0001
0AC70201 10261003 92011028 58E01028 02021029 10195080 101806F0
44001022 58E0F0E4 0A0007FF 91901002 4710F08E 0A079101 10154780
10159102 1026078E 47F000A9 10101107 078E1200 4780F0F4 19F007FC
0A0007FF 00000000 00000000 00000000 F2C7C1FA FAF7F6F5 E4FEC3F1
91C8D1CF 03BB3AB

TXT
000110 008 0001
A39AE338 0318170B

END 404040
SSERV: Performs the print and punch functions for the source statement library. The source statement library is divided into parts called sublibraries. There is one for COBOL called 'C' and one for the assembler called 'A' and in the case of OSAIA one called 'P' for cataloged JCL Books.

Format of control card:

```
@ operation @ operand1 [ , operand2]
```

Operation:

- DSPLY
- PUNCH
- DSPCH

Operand: sublib.book

sublib.ALL

ALL

Example: Print and punch the assembler sublibrary of the source statement library.

```
//JOB SORLIB
// EXEC SSERV
@ DSPCH A.ALL
/*
/*
```

8-10
* $$*/
// EXEC LNKEDT
// ASSGN SYS006,X'CRE'
// ASSGN SYS005,X'PRO'
// EXEC
CSM147206
SGM145204
1SG137193
MSG135191
SP7118181
SFC116179
SB6108168
SSG106166
SP5105156
SGT103154
SP4082136
PFC080116
PV207116
PV1067116
* $$*/
255656008FOX HENRY E 0101SGM101639264
32636478ISSAACS JUDY I 0101SSG060924487
353120375BERGERT KENNETH B 0101PV1050161948
545508335CROMBAUGH SUSAN C 0101SGM061676629
631608320ESSELBORN RAYMOND 0101CPL080514535
631985218KAZACOFF KOTCHO K 0101SGM051557952
639116773GRIEBELBAUER JOHN 0101SP6001041329
649788627JOHNSON SAM J JR 0101SGT050773699
787425848LEWIS FLOYD L 0101SGC111139387
846775121ADOMATIS RUSSELL 0101E4-130614992
848521418DEZUTTI LOUIS D JR 0101SSG030963552
888124925HOLTZMAN SALLY H 0101PFC040369948
251745252QUINCY WALTER Q 0102S061011054487
255973250OXLEY MORRIS O JR 0102PFC000206662
356879414MCCARTHY MARY M 0102SP6011054487
547083118SMITH PITTMAN S 0102PFC000206662
637427078UNDERWOOD DEVIL H 0102SP6081005936
63747514VERMILLION BETTY V 0102PV2010213992
773137376IVERA FRANCES R 0102SGT090728487
786214360WHEELER GREGORY W 0102PV1000058436
831195644TOMLINSON HARRY T 0102PV1020303236
849238650NEWTON DWIGHT N 0102SSG031019192
846493958PARKER CLYDE B 0102E5-1100832938
248041032XAVER RUTH X 0102PV2040435336
291777069ZUCKERBURG HARRY Z 0102SSG131051363
353865446YATES CLARENCE Y 0103PV1050025653
115678395GRIFFIN ARCHIE G 0103PV110010593
161312570COSTELLO LOUIS C 0103SP6131040636
337255981DIRKSON EVERETTE D 02A0CPL000405288
535973792ABBOTT BUD A JR 02A0SSG131051363
569264882FORD GERALD F 0103PV1050025653
737745671HUMPHREY HUBERT H 0103PV110010593
552
778712763 EAGLETON THOMAS E 021SGT080624264
825391253 ISLEY BROTHERS I 0201E-2010364888
923948125 BUTZ EARLE B 0201E-7071187635
263233736 JOHNSON LYNDON J 0301PV1090206487
357189154 MANSFIELD MIKE M 0301SFC011117629
66588495 KISSINGER HENRY K 03011SG021665787
762042553 NIXON RICHARD N 0301SSG050852753
297864442 LINCOLN ABRAHAM L 0301MSG101573938
384353898 QUEENIE PATRICIA 0302SP5020646935
437293380 VERDRIVE BACHMAN 0302SP6070902498
654619466 PATTON GEORGE P 0302SGT020898236
797833538 BUMSFELD DONALD R 03020PL160482498
062535910 STEPPENWOLFE CRAZE 0303SP4010465745
297275429 TRUMAN HARRY T 030PFC000361248
196468460 UNDERDOG CANINE U 0304PV2090315236
258492158 ZIMMERMAN JAMES Z 0304SSg130989953
328339176 WALLACE GEORGE W 0304PV2120025523
384422357 XAVIER I A 0304SGT070745888
611989829 YANKEE CONNECTICUT 0304SP4080448142
846244754 VICTORIOUS NOT SO 0304PV1050331888

BK END
CATALOG NEW MEMBERS TO LIBRARIES

CORE IMAGE LIBRARY: We have already seen in our discussion of Job Control that the LINKAGE EDITOR is the program that catalogs phases to this library.

EXAMPLE: Catalog an object deck to the core image library and call the new phase CATX.

```plaintext
// JOB CATLEX
// OPTION CATAL
% PHASE CATX,*
% INCLUDE

OBJECT DECK

/*
 // EXEC LINKEDT
&
```

RELOCATABLE LIBRARY: The program that catalogs modules into this library is called MAINT.

Format of control card:

```
% CATALR modulename [, v.m]
```

**Modulename** A unique name consisting of 1 to 8 alphameric characters the first of which is alphabetic.
v.m This optional entry is used to provide a version and modification number to further identify this module from other previous copies with the same name.

v can range from 0 to 127.
m can range from 0 to 255.

EXAMPLE: Catalog an available object module to the relocatable library and call it MOD516. For this copy use this month (June) for the version and today's date (27th) for the modification number.

// JOB CATALREX
// EXEC MAINT
\$ CATALR MOD516, 6.27

OBJECE MODULE

/*
/&
SOURCE STATEMENT LIBRARY: The program that catalogs to this library is called MAINT.

Format of control cards:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CATALS</td>
<td>Sublib.bookname [ ,v.m. [ , C]]</td>
</tr>
<tr>
<td>Sublib</td>
<td>The single alphabetic character which identifies which sublibrary this book is to be cataloged into.</td>
</tr>
<tr>
<td>bookname</td>
<td>The unique name for this book within its sublibrary. 1-8 alphameric character the first of which must be alphabetic.</td>
</tr>
<tr>
<td>v.m.</td>
<td>Further identification to allow us to differentiate this from previous versions.</td>
</tr>
<tr>
<td>C</td>
<td>This character is entered as part of the directory and, if used, requires the user to specify the correct version and modification number if he is trying to change the module using the UPDATE feature of MAINT.</td>
</tr>
</tbody>
</table>
This control card must appear in front and behind of each book that is cataloged into the source statement library. The optional operand entries, if used, need only be put in the front BKEND card.

- **Sub.book** same as operand in CATALS card
- **SEQNCE** The sequence numbers punched in card-columns 76 to 80 are checked for ascending sequence numbers.
- **Count** used to specify the number of cards in the book including the two BKENDs.
- **CMPRSD** compress card formats by removing blanks.

**EXAMPLE:** Catalog into the source statement library a standard file description book and call it STDFD74.

```plaintext
// JOB CATALGS
// EXEC MAINT
& CATALS C.STDFD74
& BKEND C.STDFD74 ,,,CMPRSD

SOURCE BOOK

& BKEND
/*
/8

8-46 CSD.SFW.05```
RENWING MEMBERS OF LIBRARIES

The program MAINT performs this function for all three libraries.

Core Image Library

// JOB EXAMCIL
// EXEC MAINT
PROC RENAMC oldphasename, newphasename
/*
/&

Relocatable library

// JOB EXAMRLL
// EXEC MAINT
PROC RENAMR oldmodulename, newmodulename
/*
/&

Source Statement library

// JOB EXAMSSL
// EXEC MAINT
PROC RENAMS sublib.oldbookname, sublib.newbookname
/*
/&

NOTE: All of these operations could have been done in a single job.

2. The operand field can be repeated to specify more than one rename operation.
DELETING MEMBERS FROM LIBRARIES

The program MAINT performs this function for all three libraries.

CORE image library

// JOB EXDELCIL
// EXEC MAINT
Ø DELETEC phasename

or
Ø DELETEC prog.ALL

/*
/&

NOTE: prog.ALL will delete every phase in the core image library whose name begins with the 4 characters specified.

Relocatable library

// JOB EXDELRLL
// EXEC MAINT
Ø DELETR modulename

or
Ø DELETR prog.ALL

or
Ø DELETR ALL

/*
/&

NOTE: prog.ALL deletes all modules whose first 3 characters are specified.

ALL deletes the entire library.
SOURCE STATEMENT LIBRARY

// JOB EXDELSSL

// EXEC MAINT

Ø DELETES Sublib.bookname

or Ø DELETES sublib.ALL

or Ø DELETES ALL

/*

/&

NOTE: Sublib.ALL deletes the entire sub-library

ALL deletes the entire library.
CONDENSING LIBRARIES

If a delete is performed the reference to the member in the library directory is removed. However, the old copy still occupies space in the library itself.

The same is true if a new member is cataloged with the same name as an old member. The old member is deleted from the directory but still occupies space in the library itself.

The program MAINT perform the condense function for all three libraries.

// JOB EXCONDLB
// EXEC MAINT
✓ CONDS CL
or ✓ CONDS RL
or ✓ CONDS SL
or ✓ CONDS CL, RL, SC
/*
/&
UPDATE (Change) SOURCE LIBRARY BOOKS

Once books are cataloged to the source statement library there is no need to maintain them in punched card format. However, there may be a requirement to make corrections or minor changes to library members. One way to do this is to punch up a new copy with changes and catalog it over the old copy. Another method would be to use the UPDATE function of the program MAINT to make these minor changes.

NOTE: In order to update a book it must have been sequenced prior to being cataloged. In the book card columns 73 - 76 will contain the first four characters of the bookname and card columns 77 - 80 will contain ascending sequence numbers.

The control cards are as follows:

// EXEC MAINT

WHERE UPDATE sublib.bookname, [s.book], [v.m], [nn]

) ADD seq-no

) REP first-seq-no [ ,last-seq no]

) DEL first-seq-no[ , last-seq-no]

) END [v.m.\{C\}]
UPDATE sublib.bookname ,[s.book] ,[v.m.], [mm]

Sublib.bookname  The name of book being updated
s.book  The temporary name given to the old book to preclude deleting it, in the event that an error was made in the update.

NOTE: If the update was successful this temporary book would have to be deleted.

v.m.  An optional that would be required if the old book requires change level verification.

mm.  This is the interval to be used when new book is assigned sequence numbers. Default is ___01___ if no resequence desired enter "NO" in this position.
Seq-No applies to the sequence number in card columns 77 - 80 of member book cards.

**ACTION:** The 'seq-no' in the ADD card serves to indicate behind which card in the oldbook to place the cards that follow.

**EXAMPLE:**

**OLD BOOK**

<table>
<thead>
<tr>
<th>Filler</th>
<th>Pic X (10)</th>
<th>SDFD0120</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value spaces.</td>
<td>SDFD0130</td>
</tr>
<tr>
<td>Filler</td>
<td>Pic X (9)</td>
<td>SDFD0140</td>
</tr>
<tr>
<td></td>
<td>Value 'LAST NAME'.</td>
<td>SDFD0150</td>
</tr>
</tbody>
</table>

**CONTROL CARDS**

```// JOB UPDATEIT
// EXEC MAINT
UPDATE C.SDFD150,,10
```  
```)
ADD 0130
```  
```05 Filler Pic X(4)
```  
```Value 'Dept'.
```  
```)
END
*/
&
```

**NEW BOOK**

<table>
<thead>
<tr>
<th>Filler</th>
<th>Pic X(10)</th>
<th>SDFD0120</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value spaces.</td>
<td>SDFD0130</td>
</tr>
<tr>
<td>Filler</td>
<td>Pic X(4)</td>
<td>SDFD0140</td>
</tr>
<tr>
<td></td>
<td>Value 'Dept'.</td>
<td>SDFD0150</td>
</tr>
<tr>
<td>Filler</td>
<td>Pic X(9)</td>
<td>SDFD0160</td>
</tr>
</tbody>
</table>
) REP first-seq-no [, last-seq-no]

ACTION: The cards in the old book starting with 'first-seq-no' and ending with 'last-seq-no' are deleted and the source card(s) that follow the ) REP card are put in their place. One or more old cards can be replaced by one or more new cards.

) DEL first-seq-no [, last-seq-no]

ACTION: If only the first operand is used, only that card is deleted. If the two operands are used the series of cards beginning with the first and ending with the second will be deleted.

) End [v.m. [,C ]]

This card indicates the end of updates to a book.

v.m new version and modification
	number to be placed on new book. Suggestion: month.day (6.29)

C indicates change level verification is required before any subsequent updates to this book.
REALLOCATION FUNCTION

The reallocate function is used when it becomes necessary to change the size of the libraries on the system resident pack.

// JOB CHNGIT
// EXEC MAINT

ALLOC id=cylin (track) [, id=cylin (track),....
/*
/&

id this tells the system which library CI (Core image), RL(relocatable), or SL (Source statement).

cylin Specifies the number of cylinders to be allocated for a particular library.

track Specifies the number of track to be taken from the beginning of the first cylinder to store the library index.

NOTE: for the computation of required space see IBM system control and service manual.
COPY FUNCTION

The copy function performs the following operations, individually or in combination:

1. Defines and/or creates a new system pack.
2. Defines and/or creates private libraries.
3. Transfers phases, modules, or books between any assigned libraries.

Define and create new system pack

// JOB COPY
// ASSGN SYSO02, X'191"
// ULLU IJSYSRS, 'DOS SYSTEM RESIDENCE FILE', 99/365, SD
// EXTENT SYSO02,111111, 1, 000,00001, 1219
// EXEC CORGZ

& ALLOC CL=60(10), RL=30(10), SL=30(10)
& COPY ALL

/*
*/

CORGZ The program phase that performs the copy function.

ALLOC Same as described in Reallocate function.

COPY ALL Cause all of libraries to be copied onto the new pack.
EXERCISES

1. Write the job control language necessary to print the directories of the core image library and the source statement library.

2. Write the job control language necessary to cause the following to be printed.

```
0    P.FDMS57  V.M    1.5   2 BLOCKS  SOURCE STATEMENT LIBRARY 0
0    BKEND    C.FDMS57
0    00001   RECORD CONTAINS 80 CHARACTERS
0    00002   LABEL RECORDS ARE OMITTED
0    00003   DATA RECORD IS IT01-CARD-RCD.
0    00004   01 IT01-CARD-RCD.
0    00005   05 IT01-NAME   PIC X(20).
0    00006   05 IT01-SSAN   PIC 9(9).
0    00007   05 IT01-GRADE   PIC X(2).
0    00008   05 IT01-ADDR   PIC X(20).
0    00009   05 IT01-PT-SCORE PIC 9(3).
0    00010   05 IT01-PHONE   PIC 9(4).
0    00011   05 IT01-UNIT   PIC X(10).
0    00012   05 FILLER   PIC X(12).
0    BKEND
```
3. Write the job control language necessary to catalog an object module in card deck form into the proper library and call it ZOOM5.

4. Write the job control language necessary to catalog a set of JCL cards into the proper library and call it PROC97.

5. Write the job control language necessary to delete GOPHASE from the core image library and MOD84 from the relocatable library and also condense all three libraries.
1. INTRODUCTION. Whatever the specific uses of a data processing system, certain unique operations exist that must be performed frequently. These operations may differ in detail, depending on the particular machine configuration and data format for the individual user, but the essential function remains the same. Generalized routines designed to satisfy specific functions would ease the burden of programming these operations. These routines must be flexible enough to allow the user to assign the specifications of his particular problem.

One type of program that meets these requirements are utility programs. They are designed to assist the user in day-to-day operation of his installation. With these programs, the user can perform certain frequently required operations, such as transferring disk storage files to cards or tape and printing out areas of tape or disk for program testing purposes, without programming effort.

2. EXPLANATIONS OF THE UTILITY SPECIFICATIONS.

   a. File to File Utilities. Eleven utility programs are provided for the transfer of data files from any of the normal input devices to any of the normal output devices. A file can be transferred between unlike storage mediums (tape to disk), like mediums (tape to tape), or, in the case of disk to disk, the files may be transferred from one area to another area of the same unit.

   A file can be transferred from one input medium to an output medium with these option:

   * COPY - indicates that the file is to be transferred from an input medium without change to the format of the records or file.

   * REBLOCK - transfers the input file from an input medium to an output medium with only the block size being changed.

   * FIELD SELECT - rearranges or drops fields within each input record, or converts them to zoned or packed decimal.

   * REBLOCK AND FIELD SELECT - is a combination of the reblock and field select options.

Printer output allows you to show the output in three ways:

   * DISPLAY - allows you to display a byte for byte representation of the information.

   * LIST - gives an edited representation of the information.

   * LIST AND FIELD SELECT - is a combination of the list and field select options.
For the card to printer and/or punch programs, two other output options are:

* BOTH PRINT AND PUNCH - is a combination of copy and list.

* BOTH PRINT AND PUNCH WITH FIELD SELECT - combines copy and list with field select.

These programs will handle fixed-length, variable-length, and undefined-length records, however, only fixed or variable length records can be reblocked or field selected.

b. Label Checking. Tapes containing standard labels or no labels at all can be processed without providing a user routine. When using a file to file utility program to process nonstandard or user labels or when no label checking is desired, an UPSI job control card is required. Only the 0 and 2 bits are normally used. When bit 0 is OFF, standard labels are checked on input and when it is ON, nonstandard or no label input is checked. Bit 2 OFF indicates output standard labels while bit 2 ON indicates nonstandard are not labeled.

When an UPSI card is supplied to a program, the byte is propagated from job step to job step unless another UPSI card is supplied to reset the bits. All of the UPSI bits are set to 0 following each job performed unless a new statement is supplied. When right most bits are not set by an UPSI statement, they are assumed to be zero.

c. Multifile Volume (tape). The utility programs may be used to build-multifile volumes and read from them at later dates. File positioning will be performed by logical IOCS only if the files are labeled with standard labels. Output files, nonstandard labeled files and unlabeled files are not acceptable. The file-name, volume-sequence, and file-sequence numbers must be placed in the TBL card. When using the utility programs to process multifile tape input volumes, the non-rewind-option (IN) parameter found in the utility modifier statement must be specified.

d. Multivolume Files (tape). Input or output files to these programs can consist of multiple volumes and must belong to the same data files. The control statement entries used to process the first volume are used to process each successive volume because the same fields are checked in each volume. Each tape reel of a multivolume tape file is unconditionally rewound and unloaded if no alternate tape drive has been assigned. In all other cases, the volume will be treated as specified by the input or output parameter in the utility modifier statement.

When alternate tape drives are specified and processing is completed on a particular file, the last drive processed will become the primary drive. If a new job is executed at this time, the last drive processed will then become the primary drive unless a reassignment of tape drives is made.

e. Record Skipping. Any number of logical records (up to 99,999,999) may be bypassed before processing is performed. This number can be indicated in the Rx parameter of the utility modifier statement. The number indicated
in the parameter will be the first record to be processed. Record skipping
cannot be performed for the copy function ( TC), and if specified, it will be ignored.
To skip records at the beginning of a file and copy the remainder,
the reblock function ( TR) must be indicated, and the input-description and
output-description parameters must contain identical values.

f. Sequence Numbering. Sequence generation on card output can be
indicated in the utility modifier statement. A field up to ten characters
long can be punched into each card. This field is numbered starting from 1
(with high-order zeros) and is increased by 1 for each succeeding card. If a
sufficiently long field is not defined to number all of the cards, the number
wraps around to zero without an error indication. The sequence number
overlays any data selected into the sequence area of the card. Sequence
checking can also be performed for card input to assure ascending sequence of
the specified field. If a card is out of sequence, a message is written on
SYSLST and processing continues.

g. Binary Records. When processing cards punched in column binary
format, the input and output parameters ( A and B in the utility modifier
statement) must be specified as twice the number of card columns used.

h. Printer Output. Printer output can be 120, 132, or 144 characters
per line depending on the printer used and the output format. Printer output
can be data display or list format.

* DATA DISPLAY - this format provides a visual picture of the data file.
Fixed, variable, and undefined records can be handled, but the field
select option cannot be used. Every byte of data in the file appears
in the printed output. If data display is specified ( TD), 120-
character line is forced. Only portions of the print line are used for
data. This first twenty (1-20) are reserved for information describing
the file, such as block size, block number, and record number. Data is
normally displayed in hexadecimal form, but may be optionally displayed
in alphabetic form. A heading line can be printed. A scale line is
printed at the top and bottom of each page. If record length is
specified as fixed or variable length, each logical record starts on a
new line. The input block size is printed only if the input length is
not equal to the specified block size. The excess is not printed when
the maximum length block is exceeded. Single spacing is used between
lines of print.

* DATA LIST - the data list format provides a simple edited listing of
the file. The entire print line is available for data output; output
is restricted to one line per logical record. The input record length
cannot exceed the size of the print line. If so, the field select is
required. Fields can be selected to unpack, convert to hexadecimal,
and format the page. Data list mode allows only character printing
unless a hexadecimal field is selected through a field select entry.

i. Utility Modifier Statement. This statement is used to describe the
input file that is to be processed and the output file that is desired. If
the statement is present and optional parameters are left out, default values
are assumed.

The general format of the Utility Modifier Statement is:

// Uxx Tt,Ff,A='input),B=(output),E=(c), Ix,Ox,Px,Q=(x,y),Rx,Sx

j. Field Select Statement. With this option, a field is each input
record can be moved to a different location in the corresponding output
record. Those areas of the output record that are not filled with selected
fields are filled with blanks.

When field select is used, only those bytes in the input record that are
selected will be transferred to the output record. It is therefore possible
with field select to ignore certain fields and have them dropped from the
output record.

The field select control statement provides the information for the file-to-
file programs to transfer fields from an input record to the same or a differ-
ent relative location of the output record. Each card should begin with
//FS. As many field select statements as necessary may be used and must be
complete on one statement. For example:

// FS 1,15,1/16,(P,5,3),16/72,5,19
// FS 21,2,30

The format and contents of the field select statement are:

// FS,i1,o/,i1,o/,i1,o,

i - indicates the starting position of
input record.
1 - indicates the length of field.
o - indicates the starting position of output
record.
, - separates the entries in the parameter.
/ - separates selected fields.
identifies this as a field select control
statement.

k. Heading Statement. A heading line can be printed for programs with
printed output. Heading lines are ignored if first-character forms control
is specified. A maximum of two statements can be used to indicate the
heading line desired, but the second statement need not be entered if the
first statement contains all of the desired information. The first statement
is entered // HS (followed by the information to be printed in print
positions 1-74. The second heading statement is // H2S (followed by th.
remainder of the heading line). Heading statements must follow any field
select statements used.
1. End Statement. This statement must be the last of the utility control statements in the program and is punched // END.

3. FILE TO FILE UTILITY PROGRAMS:

a. Card to Disk. The card to disk program transfers the contents of a card file from cards to an area of disk. The cards may be punched in EBCDIC or binary. The input records must be fixed-length unblocked, and each logical record must fit on one card. The maximum size input record is 80-bytes or 160-bytes for binary. These files may be copied, reblocked, field selected, or reblocked and field selected.

b. Card to Printer and/or Punch. Input records to this program must be fixed-length and unblocked. Card input and output can be either EBCDIC or binary except for both, printing and punching when it must be EBCDIC. The maximum record size is 80-bytes for EBCDIC and 160-bytes for binary.

   (1) Card to Printer. The card to printer program can produce printed output in two formats: Display and List. Sequence checking is performed on the input.

   (a) Display. This option transfers to contents to a card file to a printer with each record being placed on one print line. The field select option cannot be performed with display. In this format, the first 20 positions of the print line are reserved for information describing the file. When hexadecimal printout is called for, the entire card is printed on two lines.

   (b) List. The input records for this option are transferred to the printer with each record being fully printed. The field select option may be used. The full print line is available for printing. When hexadecimal printout is called for, the output record size is bound by the size of the print line.

   (2) Card to Punch. Records may be copied or field selected. Sequence fields are generated, but input is not checked.

   (3) Card to Printer and Punch. This program produces printed output in the list format and a reproduced card deck. Sequence fields are generated, but input is not checked.

c. Card to Tape. The card to tape program transfers the contents of a card file to tape. The cards may be punched in EBCDIC or binary. The input records must be fixed-length unblocked and each logical record must fit on one card. The maximum record size is 80-bytes for EBCDIC or 160-bytes for binary.

d. Disk to Card. The disk to card program transfers the contents of a disk file to a card file. The output file may be punched in either EBCDIC or binary. Each logical output record must fit on one card. Input records to this program must be fixed-length and may be blocked. Files in this program may be copied, reblocked, field selected, or reblocked and field selected.
All blocked input records must be reblocked.

e. Disk to Disk. The disk to disk program transfers a file between disk units or between areas of the same unit. Using the same device for input and output can cause a reduction in performance. Files can be copied, reblocked, field selected, or reblocked and field selected. If the field select or reblock options are used, the input records must be fixed or variable length.

f. Disk to Printer. The disk to printer program can display a disk file in either data display or data list format. Data display provides a visual picture of the data where every byte appears in the printed output; this format can handle fixed, variable, and undefined records. Data list provides a simple edited list of the file; input records must be fixed or variable length and (for the list function only) must not exceed the size of the print line. An option is available to this program to specify the number of logical records in a file to be bypassed before printing begins.

g. Disk to Tape. The disk to tape program transfers a file from one or more disk units to one or more tape reels. These files may be copied, reblocked, field selected, or reblocked and field selected. If the reblock or field select options are used, the input records must be fixed or variable length.

h. Tape to Card. This program transfers a tape file to a card file. The output file may be punched in either EBCDIC or binary. Input records must be fixed length. The files may be copied, reblocked, field selected, or reblocked and field selected. Blocked input records must be reblocked.

i. Tape to Disk. The tape to disk program transfers a file from one or more tape reels to a maximum of 'N' disk units where 'N' is the number of disk units assigned. These files may be copied, field selected, reblocked, or reblocked and field selected. If the field select or reblock options are used, the input records must be fixed or variable length.

j. Tape to Printer. The tape to printer program can display a tape file in either data display or data list format. Data display provides a byte-for-byte representation of the data file where every byte appears in the listing; this format can handle fixed, variable and undefined records. Data list provides a simple edited representation of the file; input records must be fixed or variable length and (for the data list function only) must not exceed the size of the print line. The field select option may be used. An option is available to this program to specify the number of logical records in a file to be bypassed before printing begins.

k. Tape to Tape. The tape to tape program transfers a file from one or more reels to tape to one or more other reels. These files may be copied, field selected, reblocked, or reblocked and field selected. Input records must be fixed or variable length if the reblock or field select options are used.
4. PHASE NAMES FOR UTILITY PROGRAMS. The phase names listed below are for the utility programs described above.

<table>
<thead>
<tr>
<th>PHASE NAME</th>
<th>PROGRAM DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDDK</td>
<td>Card to disk</td>
</tr>
<tr>
<td>CDPP</td>
<td>Card to printer and/or punch</td>
</tr>
<tr>
<td>CDTP</td>
<td>Card to tape</td>
</tr>
<tr>
<td>DKCD</td>
<td>Disk to card</td>
</tr>
<tr>
<td>DKDK</td>
<td>Disk to disk</td>
</tr>
<tr>
<td>DKPR</td>
<td>Disk to printer</td>
</tr>
<tr>
<td>DKTP</td>
<td>Disk to tape</td>
</tr>
<tr>
<td>TPCD</td>
<td>Tape to card</td>
</tr>
<tr>
<td>TPDK</td>
<td>Tape to disk</td>
</tr>
<tr>
<td>TPPR</td>
<td>Tape to printer</td>
</tr>
<tr>
<td>TPTP</td>
<td>Tape to tape</td>
</tr>
</tbody>
</table>

5. UTILITY MODIFIER STATEMENT PARAMETER. The contents of the utility modifier statement parameter is described below.

**FUNCTION**

- **TB** - Both print and punch
- **TFB** - Both print and punch with field select
- **TC** - Copy
- **TD** - Display
- **TF** - Field Select
- **TL** - List
- **TLF** - List and field select
- **TR** - Reblock
- **TRF** - Reblock and field select

**FORMAT**

- **FF** - Fixed length
- **FV** - Variable length
- **FU** - Undefined length

**INPUT DESCRIPTION**

A=(n,m) where 'n' is record length and 'm' is block length

**OUTPUT DESCRIPTION**

B=(n,m) where 'n' is record length and 'm' is block length

B=(p) where 'p' is normally 132 for a 1403N1 printer

**INPUT MODE**

- **11** - EBCDIC
- **12** - Binary
- **IN** - Do not rewind before or after data transfer (tape only)
INPUT MODE - continued

IR - Rewind before and after data transfer (tape only)
IU - Rewind before and after and unload (tape only)

OUTPUT MODE

01 - EBCDIC (punch only)
02 - Binary (punch only)
ON - Do not write disk check (disk only) or do not rewind before or after data transfer (tape only)
OC - Character (printer only)
OR - Rewind before and after data transfer (tape only)
OU - Rewind before and rewind and unload after data transfer (tape only)
OX - Hexadecimal (printer only)
OY - Write disk check (disk only)

SEQUENCE NUMBERING

Q=(x,y) where 'x' is first position of a field for sequence numbering (1 or 2 digits) and 'y' is length of field not to exceed 10 positions

PAGE NUMBERING

PN - Do not number pages
PY - Number pages

FIRST RECORD

Rx - where 'x' is the first record to be processed, normally 1

SPACING

S1 - Single space or stacker select pocket 1
S2 - Double space or stacker select pocket 2
S3 - Triple space or first character - stacker control
SA - These
SB - codes are
SC - not normally
SC - used

6. COMMONLY USED UTILITY MODIFIER STATEMENT PARAMETERS. The list shown below are the most frequently used utility modifier parameters.

```plaintext
// EXEC CDDK  (card to disk)
// UCD TR,FF,A=(80,80),B=(80,1680),E=(2314),R1,OY

// EXEC CDPP  (card to card)
// UCP TC,FF,A=(80,80),B=(50,80),I1,J1,R1,S1
```
// EXEC CDPP  (card to print double space)
// UCP TL,FF,A=(80,80),B=(132),II,0C,PY,S2,R1

// EXEC CDPP  (card to card and print single space)
// UCP TB,FF,A=(80,80),B=(80,80),II,0C,PY,R1,S1

// EXEC CDTP  (card to tape)
// UCT TR,FF,A=(80,80),B=(80,1600),II,OR,R1

// EXEC DKCD  (disk to card)
// UDC TR,FF,A=(80,1680),B=(80,80),R1,S1,E=(2314)

// EXEC DKDK  (disk to disk)
// UDD TR,FF,A=(80,1680),B=(80,1680),E=(2314,2314),OY,R1

// EXEC DKPR  (disk to print double space)
// UDP TL,FF,A=(80,1680),B=(132),E=(2314),PY,OC,S2,R1

// EXEC DKTP  (disk to tape)
// UDT TR,FF,A=(80,1680),B=(C0,1600),E=(2314),R1,OR

// EXEC TPDC  (tape to card)
// UTC TR,FF,A=(80,1600),B=(80,80),II,01,R1,S1

// EXEC TPDK  (tape to disk)
// UTD TR,FF,A=(80,1680),B=(80,1680),E=(2314),IR,OY,R1

// EXEC TPPR  (tape to print double space)
// UTP TL,FF,A=(80,1600),B=(132),II,PY,R1,S2

// EXEC TFTP  (tape to tape with field select)
// UTT TRF,FF,A=(80,80),B=(81,61),II,OU,R1

7. For additional information on the utility programs, refer to IBM SRL GC 24-3465, DOS and TOS Utility Programs.
Job control language for COBOL PE 2 (OS)

//C23
PROC DUMP = 'DUMMY'

/* THIS PE READS NAMES AND LIST THEM*/

/* CREATED ON 30 NOV 76 BY SOFTWARE DIV, CSD, USAIA*/

//COB
EXEC PGM=IKFCBL00

// PARM = 'LOAD, SUPMAP, SIZE=84K, BUF=8K, NODECK, CLIST'

//STEPLIB
DD DSN=USAIA.SOFTWARE.LINKLIB, DISP=SHR

//SYSPRINT
DD SYSOUT=A

//SYSPUNCH
DD DUMMY

//SYSLIB
DD DSN=USAIA.SOFTWARE.SOURCE, DISP=SHR

//SYSLIN
DD DSN=LOADSET, DISP=(MOD, PASS), UNIT=SYSDA,
// SPACE=(80(500,100))

-------------- COBOL source deck would be placed here ------------

//LKED
EXEC PGM=IEWL, PARM = 'LIST, XREF, MAP, LET', COND = (8, LT, COB)

//STEPLIB
DD DSN=USAIA.SOFTWARE.LINKLIB, DISP=SHR

//SYSLIB
DD DSN=LOADSET, DISP=(OLD, DELETE)

//SYSLMOD
DD DSN=COSET(COB), DISP=(NEW, PASS), UNIT=SYSDA,
// SPACE=(1024(50,20,1))

//SYSLIB
DD DSN=SYS1.CORLIB, DISP=SHR

//SYSLIB
DD UNIT=(SYSDA, SEP=(SYSLIN, SYSLMOD)), SPACE=(1024(50,20))
//SYSPRINT DD SYSOUT=A
//GO EXEC PGM=*.*ED.*YSLMOD,COND=((8,LT,C0B),(4,LT,.*ED))
//SYSOUT DD SYSOUT=A
//SYS005 DD SYSOUT=A
//SYS006 DD DSN=USA1A.SOFTWARE.DATASET.C9,DISP=SHR
OS JCL to execute a COBOL PE

//name
JOB (procname, room), 'programername', MSGLEVEL=(1,1)
//
EXEC procname

//COB.SYSIN
DD *

COBOL

source

deck

/*
//

name
This is a name assigned to the job. This parameter is taken from the first eight characters of your name on the 'buff colored' control card you now put on your deck.

procname
Use the code from the buff colored card C11, C09, C23, ...

room
Enter your classroom number

programername
'John Q. Smith'

procname
Same as above.
START

// JOB name

PERFORM OPTION ROUTINE

TO BE CATALOGED?

YES → PHASE CARD

NO

TO BE COMPILED?

YES → // EXEC FCOBOL

NO

LINK REQUIRED?

YES → PERFORM LINK EDIT ROUTINE

NO

TO BE EXECUTED?

YES → PERFORM EXECUTE ROUTINE

NO

STOP
OPTION ROUTINE

// OPTION ...

LIST JCL

YES ➞ ....LOG

NO

PGM CATALOGED ?

YES

TO BE CATALOGED ?

YES ➞ ....CATAL

NO

LINK REQUIRED ?

YES ➞ ....LINK

NO

TO BE COMPiled ?

YES ➞ COMPILER OPTIONS

NO

TO BE EXECUTED ?

YES ➞ ....DUMP

NO

EXIT

SYM XREF ERRS DECK

LIST LISTX

A-2

CSD.SFW.OS
EXECUTE ROUTINE

// ASSGN .... for cards and print

USE TAPE FILES?
YES

alternate

// ASSGN ....

// TLBL ....

MORE FILES
YES

NO

USE DISK FILES?
YES

multi vol

// ASSGN ....

// DLBL ....

// EXTENT ....

MORE DISK FILES?

YES

NO

PGM CATALOGED?
YES

// EXEC

// EXEC phase

CARD DECK?
YES

EXIT

CARD DECK

NO
OBJECTIVES:
1. To describe in detail the three mainframe components of the IBM SYS/360 computer, Main Memory, the Central Processing Unit, and Data Channels.
2. To describe the five machine language formats.
3. To describe interrupts and interrupt processing.
4. To define PSW, IOCS, LIOCS, and PIOCS.

I. MAIN MEMORY. - Where data and instructions must ultimately be to be operated upon by the system.
   A. Byte -
   B. Addressing -
   C. Concept of "Word Units"
      1. Fullword -
      2. Halfword -
      3. Doubleword -
      4. Boundary Alignment

II. CENTRAL PROCESSING UNIT (CPU).
   A. Instruction Cycle.
      1. I-Time -
      2. D-Time -
      3. The Difference Between Instructions and Data -

IA-01-01-05
B. Access Width -

C. Access Time -

D. Control Element - Fetches and decode machine instructions and sets up internal logic paths for the actual execution of the instruction.

E. Arithmetic and Logical Unit (ALU) - Responsible for actually executing the instructions as decoded by the Control Element.

1. General Purpose Registers (GPR).
   a. Size and Number -
   b. Data Format -
   c. Uses
      1) 
      2) 

2. Floating Point Registers (FP REG).
   a. Size and Number -
   b. Data Format -
   c. Use

3. Operations.
   a. Arithmetic -
   b. Logical -
III. DATA CHANNELS - Special purpose minicomputers whose only function is to monitor I/O operations.

A. Modes of Operation.
   1. Burst -
   2. Multiplex -

B. Types of Channels.
   1. Selector -
   2. Multiplexor -

C. Channel Command Word (CCW).

D. Characteristics of Channels.
   1.
   2.
   3.
IV. THE MODULAR CONCEPT OF THE SYS/360.

V. MACHINE LANGUAGE INSTRUCTION FORMATS.

A. The Two Theories of Machine Language Instruction Formats.

1. Fixed Length
   a.
   b.
   c.

2. Variable Length
   a.
   b.
   c.
B. Parts of a Machine Language Instruction.

1. Operation Code (OPCODE) - The first byte of the instruction is always the OPCODE. It defines the arithmetic or logical or I/O operation that is to be done.

2. Operands.
   a. Registers -
   b. Storage -
   c. Data -

C. Formats.

1. Register to Register (RR).
   a. Length -
   b. Operands -
   c. Diagram -
   c. Type Instructions -

2. Register to Storage (RS).
   a. Length -
   b. Operands -
   c. Diagram -
   d. Type Instructions -

3. Register to Indexed Storage (RX).
   a. Length -
   b. Operands -
4. Storage and Immediate (SI).
   a. Length -
   b. Operands -
   c. Diagram -
   d. Type Instructions -

5. Storage to Storage (SS)-
   a. Length -
   b. Operands -
   c. Diagrams -
   d. Type Instructions -

VI. Interrupts - The mechanism that allows the machine to change from the problem program state to the supervisor state.

A. Types of Interrupts.
   1.
   2.
   3.
   4.
   5.
B. Priority of Interrupts for Simultaneous Occurrences.

1. Stacking Priority.
   a. 
   b. 
   c. 
   d. 

2. Execution Priority.
   a. 
   b. 
   c. 
   d. 

3. Handling of Simultaneous Interrupts

C. Program Status Word (PSW).

1. 

2. Current PSW -

3. New PSW -

4. Old PSW -

5. Diagram -

<table>
<thead>
<tr>
<th>SYSTEM MASK</th>
<th>KEY</th>
<th>INTERRUPTION CODE</th>
<th>CPU MASK</th>
<th>INSTRUCTION ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IA-01-01-05
D. Interrupt Processing.

1.

2.

3.

4.

5

VII. Sys/360 Input/Output

A. IOCS. IOCS stands for Input Output Control System. In the supervisor there is a small block of coding which serves to initiate I/O operations in response to program requests. It also notifies your program that an I/O operation is complete.

B. PIOCS. Physical IOCS is the most basic method of accessing the IOCS section of the supervisor. In PIOCS the programmer has direct control over I/O operations but is responsible for all error checking and program notification of completed I/O.

C. LIICS. Logical IOCS is the system of accessing I/O devices most often used. LIICS provides for all error handling and program notification of completed I/O. In order to do this there are 3 parts to the LIICS system:

1. A table used to define the appearance of the file being used. This table is called a Define The File table or DTF.

2. A block of executable coding to handle error checking, I/O initiation, etc., called a module.

3. An I/O imperative specifying reading or writing on the file.
Keywords: TABULAR-MODULAR STRUCTURE, I/O AREA, DEVICE ADDRESS, MODULE NAME, PROGRAMER OPTIONS

In order for the 360 to perform an I/O operation three things must be done:

1. Specific information relating to the file must be made available in a table called a DTF table.

2. Coding (a piece of program) must be available to reference the table and pass the information to the channel. This block of coding is called an I/O module.

3. A specific command must be given to initiate the process.

The following information is needed in the table:

1. A Device Address consisting of a logical unit address must be provided. This address is then used to locate the physical address of the unit.

2. The location of the data must be specified explicitly. This area is called an I/O Area and is first defined by the programer. This area is used to hold data for output or if it is an input area to receive data. Usually there is at least one I/O area per file.

3. Any special device dependent options are also specified. These are called programer options. An example would be how a tape is blocked or whether a tape is to be rewound at the end of the job.

4. The assembler takes into account all of the above information and determines which relocatable module will do the job.

I/O Module

The I/O module is resident in the relocatable library. There are I/O modules for each set of options for each device.

The linkage editor is given the name of the appropriate module in the external address portion of the symbol table. It then pulls it off the relocatable library and links it in to your program.

The I/O module provides an interface between your program and the IOCS routines in the supervisor. It also provides for error checking and program notification of completed I/O. It also provides the coding necessary for implementation of programer options.
Imperative Command

In order to access the table and module for I/O the programmer need only use the GET (filename) or PUT (filename) commands. These commands provide branches to the appropriate I/O routines.

CPU

1. GET instruction branches to module.
2. Module obtains information it needs from DTF table.
3. Module passes information to IOCS routines in supervisor and requests I/O initiation.
4. IOCS routine passes information to channel and starts I/O.
5. Channel places data in I/O area in core.
6. Channel notifies IOCS of I/O completion.
7. IOCS passes completion information to module. The module then analyzes completion code to determine what error conditions, if any, are present.
8. If no errors, control is passed back to program.
Computer Systems Command Interface
With the Field Data Processing Activity

A Self-Paced Referenced Text

Programmer/Analyst Course
Software Division, Computer Science Department

January, 1979
OBJECTIVE: Given a situation at a field DPA running standard Army systems, be able to report an incident or a recommended system change on the proper forms and specifying the required information.

CRITERION TEST: The test for this class is different from the others in the course in several ways. First of all, the test is on a go/no go basis, that is, you either get all the points or you get none of them. Secondly, the test is designed to have you demonstrate your ability to perform the two primary CSC interface tasks. You will be given a copy of the test at the beginning (not the end) of the class time allotted to this block. Finally, the test will be graded with you present and, if it is not satisfactory, you will be told what is incorrect and asked to do the test over. This is because we believe that a test should be a learning tool as well as a demonstration of your performance.

METHOD OF LEARNING: This is a self-paced module and as such, you determine how fast you want to learn. If you have any questions on this topic, you may talk with other students or with the instructor. When you have finished the body of this text, complete the criterion test and take it to the instructor for one-to-one feedback. The only restrictions are the test must be finished by the end of the class time allotted and you may not talk with other students when taking the exam.
PURPOSE

This text was designed to acquaint you with how the Army maintains its standard software systems and the part that you, the programmer, play in the overall scheme. Speaking in more specific terms, this class will show you how to report problems with systems that were not written by you but are running at your DPA. By the way, if you do not know what that abbreviation means, there is a glossary in the back of this text that was designed for people who are not familiar with the Army's unique ADP terminology. Chances are that if you find other unfamiliar terms during this class you will find them defined in the glossary also.

WHY SOFTWARE STANDARDIZATION?

Although the Army has many different units scattered throughout the world, their data processing requirements are really quite similar. They all have, for instance, the following needs:

* Keep personnel data current and available.
* Requisition supplies.
* Keep soldier's pay data current.
* Maintain accountability of government property.

If each Army DPA had to develop software systems to accomplish these and other tasks, there would have to dozens of programmers at the DPA and lots of spare run time to compile and test programs. Additionally, people moving from Post to Post (a fact of Army life) would have to spend many hours being retrained on new systems that did basically the same tasks as the systems at their last Post. With a lack of standardization among ADP systems, attempting to send data between these systems would be a nightmare.

CSC COMES INTO THE PICTURE

To alleviate these problems, the Army designated the Computer Systems Command (CSC) as the central design agency for all ADP systems that are run at more than one command. Under direction of the functional proponent (it's in the glossary), CSC develops standard software systems and sends them to all DPA who have a requirement for them. Not all DPA run the same standard systems, (for instance: a DPA at an Infantry Division would not have the same ADP requirements as the DPA that supports Fort Harrison).

When CSC sends a system out it comes in a package containing operator's documentation, JCL, executable phases and instructions on how to load the phases on the libraries. Notice that there are no source programs or programmer documentation in the package. This is to preclude individual commands from changing the programs and therefore not having the same system as other commands. Some of the systems developed by CSC that you are likely to encounter are:

IA-01-04-26
* SIDPERS - Maintains complete personnel data and prints personnel reports. Used at all Divisions and Posts.

* SAILS - Standard supply system at Posts, Corp and supply inventory and management centers.

* STANFINS - The system that keeps track of Army spending at each Post.

* DLOGS - Used at all Divisions for requesting supplies and maintaining property records.

These are just some of the systems that CSC is responsible for. They actually have many systems varying:

* in complexity from under 10 programs to over 100.

* in Hardware from a UNIVAC 1005 to an IBM 370.

* in scope from commissary management to Army Force Accounting.

WHAT DO I DO?

At this point you may be asking yourself what could you possibly have to do with all this since all CSC systems have already been developed and made operational. The answer to this question is why you are having this class!

When you go to a permanent assignment at a DPA you will probably be responsible for assuring the correct operation of an AMIS (remember the glossary!). Your duties in this area will be to:

* Load system changes to your libraries to keep the system current.

* Troubleshoot problems occurring during the execution of an AMIS, and report problems that cannot be overcome to the Assigned Responsible Agency (ARA).

* Forward recommended changes of an AMIS to the ARA.
We will cover each of these tasks separately in the following paragraphs.

**KEEPING THE SYSTEM CURRENT**

Just as the Army requirements and standards change from time to time so do the AMIS that support the Army. CSC produces system change packages (SCP) on a regular basis, usually every 3 - 6 months, that will keep the system current with Army policy. An SCP comes with all necessary documentation and load instructions. All that is required of the programmer (you) is to forward the package to the operators with any local run instructions and review the output from the SCP load to make sure that the system was loaded correctly. Following the load, you must notify CSC of the date and time of the load. You should be sure to load the SCP as soon as possible so that your system is the current one.

Occasionally, there will be emergency requirements to change a system, usually due to a program failure. When this occurs, CSC sends an Emergency/Urgent Change Package (EUCP) to all affected DPA. EUCP are normally not mailed but are sent through the Automatic Digital Network (AUTODIN) and will arrive within 2 days of dispatch. These change packages must be loaded immediately upon receipt.

**WHAT IF THE SYSTEM DOESN'T WORK RIGHT**

If an AMIS does not work right, either from an ADP standpoint or a functional standpoint, this means that an incident has occurred. Analyze the incident to make sure it is not caused by a local problem. Once you determine that the problem is not local, report it to CSC as soon as possible by phoning the Customer Assistance Office (CAO). The information that the CAO needs can be taken directly from the Incident Report form, (USACSC Form 53) that you must fill out for your records. Let's take a look at this form in detail, line by line. Refer to the sample form while you read the items below.

1. a. Enter your post name.
   b. This is a number consisting of three fields:
      - The AMIS code
      - Your DPA code
      - A three position sequence number
        a sample number would be: CO2-R111-027
c & d. This code designates your DPA type and will be furnished you at your DPA.

e & f. Your name and phone.
g. The hours (in local time) when you work.
h. Your name (unless someone else phones the incident to CSC).

2. a. Leave blank.
b. AMIS Name.
c & d. Self-evident.
e. The person you talked to at CSC.
f. CAO phone number.

3. a. If the incident caused the job to blow-up indicate that and enter the termination code printed on the output listing.

b. Self-evident.

4. This is the number of the SCP or EUCP that was loaded most recently and the date it was loaded (installed). If the problem involves a program, enter the program name in block A or if the problem is in the system documentation enter the documentation name in block B.

5. a. Enter the job name here.

b. Explain in detail exactly what happened. This is very important because it is from this description that CSC will start to resolve the problem.

c. Explain what you did, if anything, to get around the problem.
d. What effect does this incident have on you?
e. Leave blank.

6. Most often this will be applications software.

7. Which of these items are available for CSC to refer to, if needed.

You should maintain a file of all incident reports and indicate in that file the action taken by CSC to resolve the problem.
INCIDENT REPORT

For use of this form, see USACSC REG 18-21-1; the proponent agency is Quality Assurance Directorate. (Test & Conf Mgt Div)

1A. INSTALLATION Fort Harrison - USAIA

2A. CAO ID

B. ORIGINATOR NUMBER PO2-R111-006

C. FUNCTIONAL AMIS SIDPERS

3. DATE RECEIVED 12 Jan 79

D. TIME RECEIVED 1600

4. RECEIVED BY SFC Jones

E. PHONE 354-1616

5. CONTACT SP4 Smith

F. CONTACT'S PHONE 699-2323

G. CONTACT WILL BE AVAILABLE 0730-1630

H. PERSON PHONING IN INCIDENT SP4 Smith

3. OPERATING STATUS AT TIME OF INCIDENT

A. WAS THERE A CYCLE HALT OR ABNORMAL TERMINATION IN THE PRODUCTION RUN?

[ ] YES ABEND CODE WRONG LENGTH RECORD

[ ] NO

B. DATE/TIME INCIDENT DETECTED:

1200 Jan 79, 1500

4. BASELINE P02-21-05 INSTL'D 5 Jan 79

A. PROB PGM ID P3AAAC VERS

B. PROB DOC ID

CHG LEVEL PARA(S)

PAGES

5. DESCRIPTION OF INCIDENT.

A. ID OF RUN IN PROGRESS DAILY CYCLE

B. NARRATIVE DESCRIPTION OF INCIDENT AND RELATED EVENTS Job aborted due to wrong length record on input tape P3AAAC. Listed tape and saw that all records were one byte too long.

C. ACTION TAKEN BY USER AND RESULTS

Used TPTP utility to strip the extra byte and continued.

D. IMPACT ON USER UNTIL RESOLVED

Every time the cycle is run, the tape must be reformatted.

E. CAO/ASD REDEFINITION OF PROBLEM IF DIFFERENT THAN KB.

6. TYPE OF DIFFICULTY

X APPLICATION SOFTWARE AVAIL

EXECUTIVE SOFTWARE X CORE DUMP AVAIL OUTPUT FILE-TAPE

INPUT DATA X CONSOLE COPY DATA FILE-TAPE

DOCUMENTATION X JOB CONTROLLIST RPT OUTPUT-TAPE

HARDWARE X SYSLST OUTPUT DOCUMENTATION

OTHER X INPUT FILE-TAPE OTHER

DISK MODULE

7. SUPPORTING DOCUMENTATION FOR RESEARCH BY USER

X APPLICATION SOFTWARE AVAIL

EXECUTIVE SOFTWARE X CORE DUMP AVAIL OUTPUT FILE-TAPE

INPUT DATA X CONSOLE COPY DATA FILE-TAPE

DOCUMENTATION X JOB CONTROLLIST RPT OUTPUT-TAPE

HARDWARE X SYSLST OUTPUT DOCUMENTATION

OTHER X INPUT FILE-TAPE OTHER

DISK MODULE

USACSC FORM 53

Replace edition of 1 May 75
After you have had an opportunity to observe an AMIS in its normal operation, you may think of ways to improve it. Additionally, the people who use the output, the functional user, may like to see the output changed or possibly new reports produced. CSC encourages these suggestions and the Army has provided a means of submitting them: the System Change Request (SCR). There are two types of change requests, technical, involving a suggestion for technical ADP reasons and functional, involving a suggestion to change the input or output from the customer's viewpoint. You will be responsible for filling out technical SCR and you should review functional SCR after they have been prepared by the user. You should keep a log of all SCR you have submitted. CSC distributes a monthly SCR status report which you can compare to your SCR log to make sure that your information agrees with CSC's information.

Refer to the sample SCR and we will discuss the blocks you will have to fill out.

Block 1. This will be the address of the functional proponent of the system.

Block 2. The address of your DPA.

Block 3. This number has the same format as the originator number in the incident report.

Block 4. Your name.

Block 5. You can only check routine. The other three blocks are for the functional proponent and CSC.

Block 6. The AMIS name and, if applicable, the program name.

Block 7. If this SCR is the result of an incident, then complete this block. Otherwise leave blank.

Block 8. Self-evident.

Block 9. If the SCR involves system documentation, check the appropriate block.

Block 10. Attach any documents that may help to explain the suggestion and check the appropriate block.
Block 11. Enter the problem as it currently exists in part A. In part B describe the solution you are suggesting in sufficient detail so that it can be clearly understood.

Block 12. Usually you will have to furnish a copy of the SCR to CSC and to your major command MISO.

Block 13. Your name, job and signature.

Block 14. Leave blank.
To: HQDA (DACA-CSS-F)  
HQDA, Pentagon  
Washington, DC 20310

From: MISO  
Fort Harrison, IN 46216

Originator No.: FO1-R111-016

Subject: Operator MSG in Program P05ANV

Incident Encountered: 
Station: 
Date: 
Time: 

Short Title: Operator MSG in Program P05ANV

Narrative:

Problem Description: Program P05ANV prints 5 lines of operator messages. This takes excessive console and operator time.

Recommended Solution/Action Taken:
Delete the unnecessary words and abbreviate the remainder so that only 1 or 2 lines are printed.

Copy Furnished: 
HQ TRADOC MISO 
USACSC; CSCS-QAC-M

Prepared By: SP5 Daniel Johnson  
Programmer/Analyst

PropONENT Agent Review:

Type of Change:
- Functional
- Technical

Class of Change:
- Regulatory
- Non-Regulatory

Extent of Change:
- Major
- Minor

Referred To ARA for Analysis (Date):

Disposition:
- Approved, Requested Implementation
- Disapproved

Functional Guidance:
- Attached
- Not Required

SIGNED: ______________________ DATE: ____________
WHAT NOW?

We hope that you now have an understanding of your relationship in a DPA to the Computer Systems Command. Incident and recommended change reporting will be a fact of Army life to you.
GLOSSARY

1. Army Management Information System (AMIS). This term refers to any software system used in the Army. When used in this text, however, the term refers to standardized software that is designed, tested and programmed at Computer Systems Command and run at many DPA. Sometimes this type of AMIS is called SAMIS (Standard AMIS) or Class A AMIS.

2. Assigned Responsible Agency (ARA). The agency who designs, programs and tests AMIS. CSC is normally the ARA for Class A AMIS.

3. Automatic Digital Network (AUTODIN). A communications link among most Department of Defense activities used for transmitting data from post to post. Data is transmitted electronically (similar to a phone call) and is received either in punched cards or magnetic tape.

4. Computer Systems Command (CSC or USACSC). This is the Army's software design and planning agency. It develops, programs and tests centralized Class A AMIS and plans for the future direction of Army software.

5. Customer Assistance Office (CAO). This is the central contact point that CSC has established for reporting incidents. It is generally manned 24 hours a day. The CAO will take all the information on the incident report and forward it to the responsible programmer for resolution.

6. Data Processing Activity (DPA). Any Army organization which operates data processing equipment. This can range from a unit with just data entry equipment to a unit with several large scale computer systems.

7. Emergency/Urgent Change Package (EUCP). Changes to standard AMIS sent by CSC to resolve operational difficulties. This is the way CSC distributes changes made as a result of incident reports.

8. Functional Proponent. The organization which is responsible for the functions of an AMIS. For example the Deputy Chief of Staff for Personnel is the functional proponent for SIDPERS. Normally HQDA staff offices are the functional proponents for CSC systems.

9. Functional User. The activity which uses the output from an AMIS; usually non-ADP personnel who are customers of the DPA.

10. Incident Report (IR). Reporting a system operational problem, that is, where the system does not work as it should, to CSC via telephone. This is also the written record (USACSC form 53) made in support of the phone call.
11. System Change Package (SCP). A regularly scheduled change to a standard AMIS developed by the ARA and sent to all DPA who run the affected system. An SCP will normally include changes made as the result of SCR's.

12. System Change Request (SCR). A request to either the PA or ARA of a standard system to make a change in the system. If the change is for ADP efficiency, it is called a technical SCR; if it is to change data on initial input or output it is called a functional SCR.
HARDWARE/SOFTWARE SYSTEMS EFFICIENCY

SCENARIO

1. SITUATION:

   a. As previously stated, you have been assigned to this organization as the Systems Software Analyst, responsible for the operating system and associated software functions.

   b. New equipment configuration.

      (1) IBM 360 Model 30 - 64K

      (2) 1050 Console Keyboard 1 ea

      (3) 2314 Disk w/9 drives 1 ea

      (4) 2400 Tape Drive (9 track) 4 ea

      (5) 2804 Control Unit 1 ea

      (6) 1403 Printer 1 ea

      (7) 2540 Card Read/Punch Unit 1 ea

      (8) 2821 Control Unit 1 ea

      (9) Selector Channel 2 ea

      (10) Multiplexer Channel 1 ea

   c. Special Features

      (1) Floating Point

      (2) Decimal

      (3) Timer

   d. To support

      (1) Multiprogramming

      (2) COBOL

      (3) FORTRAN

      (4) PL/I (48 character set)
e. Manufacture Supplied Packages

(1) Operating System Routines (Supervisor, Linkage Editor, etc)

(2) Subroutines for FORTRAN and PL/I

(3) Utility Programs

f. User systems now in effect are outlined in the separate systems charts, incl 1 and 2. Memory requirements and run times for each process step are indicated on the chart. Additional programs to be supported are listed in incl 3 and are to be assumed to be stand-alone.
2. REQUIREMENTS

   a. Design and generate, on paper, an operating system specifying selected options, core requirements, and total size of your supervisor. Calculate the "Send" Address.

<table>
<thead>
<tr>
<th>OPTIONS/SELECTIONS</th>
<th>SUPV</th>
<th>CONFIG</th>
<th>STDJC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPS=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TP=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MICR=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASCII=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AP=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EU=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERRLOG=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCRR=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODEL=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMER=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PORT=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECK=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIST=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LISTX=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAIN STORAGE REQUIREMENTS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options/Selections</td>
<td>Main Storage Requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYM=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XREF=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERRS=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHARSET=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOG=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUMP=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINES=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPARM=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOPT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OC=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IT=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEB=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEBV=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVA=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SKSEP=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JA=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JALIOCS=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTO=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDRA=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OLTEP=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETAIN=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCIL=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBF=</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

613
<table>
<thead>
<tr>
<th>Options/Selections</th>
<th>Main Storage Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCHAIN=</td>
<td></td>
</tr>
<tr>
<td>TRKHLD=</td>
<td></td>
</tr>
<tr>
<td>AB=</td>
<td></td>
</tr>
<tr>
<td>WAITM=</td>
<td></td>
</tr>
<tr>
<td>DASDFP=</td>
<td></td>
</tr>
<tr>
<td>SYSFIL=</td>
<td></td>
</tr>
<tr>
<td>PIOCS</td>
<td></td>
</tr>
<tr>
<td>SELCH=</td>
<td></td>
</tr>
<tr>
<td>BMPX=</td>
<td></td>
</tr>
<tr>
<td>CHANSW=</td>
<td></td>
</tr>
<tr>
<td>TAPE=</td>
<td></td>
</tr>
<tr>
<td>MRSLCH=</td>
<td></td>
</tr>
<tr>
<td>IOTAB</td>
<td></td>
</tr>
<tr>
<td>BGPGN=</td>
<td></td>
</tr>
<tr>
<td>F1PGN=</td>
<td></td>
</tr>
<tr>
<td>F2PGN=</td>
<td></td>
</tr>
<tr>
<td>JIB=</td>
<td></td>
</tr>
<tr>
<td>CHANQ=</td>
<td></td>
</tr>
<tr>
<td>IODEV=</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
</tr>
</tbody>
</table>

Send Address: SEND ____________________
b. List available channels and distribution of I/O devices:

<table>
<thead>
<tr>
<th>Type</th>
<th>Channel</th>
<th>Channel Number Assigned</th>
<th>Devices</th>
</tr>
</thead>
</table>

c. Specify size of required partitions:

ALLOC  $F1=______K, F2=______K$
d. Prepare a schedule of jobs for a Tuesday in which the payroll checks are to be run. Shift starts at 0700 hrs and we run two shifts of eight hours each. Indicate Program Number, Time to be Run and Partition. Schedule should be such that the first job on the schedule is the first job to be run, etc.

<table>
<thead>
<tr>
<th>Program Number</th>
<th>Time to be Run</th>
<th>Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0704</td>
<td></td>
</tr>
</tbody>
</table>
e. List contents of each library in specific terms, by routine name or program number.

(1) Core Image Library

(2) Relocatable Library
f. Does your system require "Spooling"? Yes _____ No _____ Why?
CONTINGENCY PLANNING

FREQ: As Required

Unit Sel

Master Type Unit File

Prog #C01 Select Unit 10K - 25 min

Type Unit Audit List

Selected Type Unit

Planners

Corrections

Selected Type Unit Equipment File

Prog #C02 Select Equip 12K - 45 min

Type Unit Equip Audit List

Planners

Changes

Equipment File

Prog #C03 Equip Modification 15K - 10 min

Modified Selected Type Unit Equip File

Modified Selected Type Unit File

1
Note 1: The errors are to be corrected and the corrections inputed into the next day's cycle, except those errors of the last cycle for the period are to be corrected immediately and the edit rerun. All errors must be corrected before the Payroll can be run.

Note 2: The errors must be corrected immediately and the edit rerun until no errors exist. Only then will the payroll checks be run.

Incl 2
### ADDITIONAL PROGRAMS

<table>
<thead>
<tr>
<th>PROG NR</th>
<th>SIZE K</th>
<th>FREQ</th>
<th>TIME REQD</th>
<th>RUN TIME</th>
<th>INPUT FILES</th>
<th>OUTPUT FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TAPE DISK READER</td>
<td>TAPE DISK PUNCH PRINTER</td>
</tr>
<tr>
<td>A01</td>
<td>27</td>
<td>Daily</td>
<td>1000 hrs</td>
<td>10</td>
<td>2 1 X</td>
<td>1</td>
</tr>
<tr>
<td>A02</td>
<td>10</td>
<td>Daily</td>
<td>0800 hrs</td>
<td>3</td>
<td>1 1 X</td>
<td>1</td>
</tr>
<tr>
<td>A03</td>
<td>33</td>
<td>Daily</td>
<td>2400 hrs</td>
<td>25</td>
<td>2 3 X</td>
<td>2</td>
</tr>
<tr>
<td>A04</td>
<td>20</td>
<td>Daily</td>
<td>2400 hrs</td>
<td>57</td>
<td>1 2</td>
<td>2</td>
</tr>
<tr>
<td>A05</td>
<td>15</td>
<td>Daily</td>
<td>1200 hrs</td>
<td>15</td>
<td>3 X</td>
<td>3</td>
</tr>
<tr>
<td>A06</td>
<td>25</td>
<td>Daily</td>
<td>2400 hrs</td>
<td>30</td>
<td>2 X</td>
<td>2</td>
</tr>
<tr>
<td>A07</td>
<td>35</td>
<td>Daily</td>
<td>2400 hrs</td>
<td>240</td>
<td>1 4 X</td>
<td>4 2</td>
</tr>
<tr>
<td>A08</td>
<td>34</td>
<td>Tuesday</td>
<td>2400 hrs</td>
<td>65</td>
<td>2 4</td>
<td>2 3</td>
</tr>
<tr>
<td>A09</td>
<td>29</td>
<td>Friday</td>
<td>2400 hrs</td>
<td>30</td>
<td>2 X</td>
<td>1 1</td>
</tr>
<tr>
<td>A10</td>
<td>15</td>
<td>Monday</td>
<td>2400 hrs</td>
<td>27</td>
<td>1 3</td>
<td>2 1</td>
</tr>
<tr>
<td>A11</td>
<td>21</td>
<td>Wed</td>
<td>2400 hrs</td>
<td>32</td>
<td>2 4 X</td>
<td>2</td>
</tr>
<tr>
<td>A12</td>
<td>17</td>
<td>Thur</td>
<td>2400 hrs</td>
<td>19</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>A07</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>A08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>A09</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>A10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>A11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>A12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

**INPUT FILES**

<table>
<thead>
<tr>
<th>TAPE</th>
<th>DISK</th>
<th>READER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**OUTPUT FILES**

<table>
<thead>
<tr>
<th>DISK</th>
<th>PUNCH</th>
<th>PRINTER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Spooling

18 If determined to be required

---

Compile and Test Time

Software Maintenance/Software Analyst

Systems Maintenance/Manufactures Rep