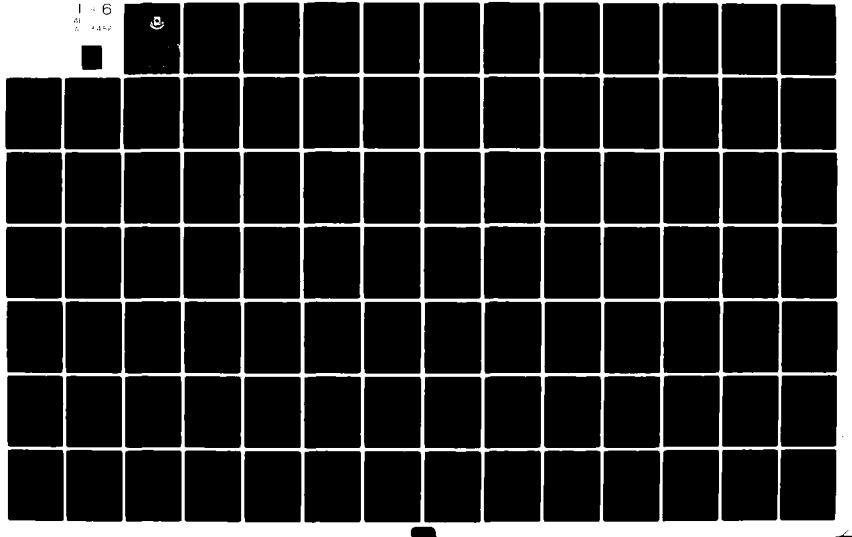
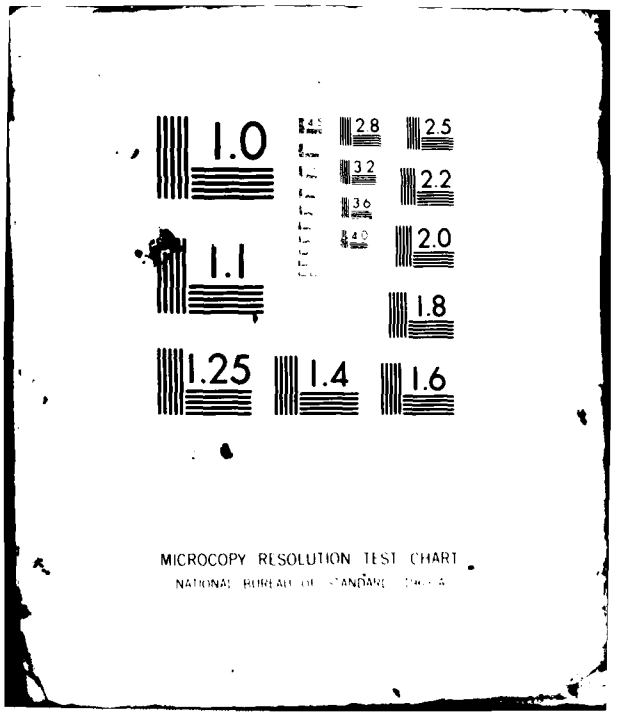
AD-A113 456 ARMY COMPUTER SYSTEMS COMMAND FORT BELVOIR VA F/6 9/2
PROGRAMING PROCEDURES MANUAL (PPM).(U)
DEC 81

UNCLASSIFIED NL




1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART

15 DECEMBER 1981

USACSC MANUAL 18-1-1

AD A113456

# UNITED STATES ARMY

# COMPUTER SYSTEMS COMMAND

DTIC
ELECTE
APR 1 4 1982
H

PROGRAMING PROCEDURES MANUAL (PPM)

DTIC FILE COPY

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

82 04 12 185

FORT BELVOIR, VIRGINIA 22060

# PROGRAMING PROCEDURES MANUAL

This manual prescribes the procedures to be used by US Army Computer Systems Command (USACSC) programers in developing and maintaining multicommand and Command-unique ADP systems.

In view of the emphasis by General Services Administration (GSA) that Federal agencies comply with Federal standards (FIPS/FED-STDS) in the procurement process, effective 1 January 1979, all COBOL compilers offered will be validated on a scheduled annual basis by the Federal Compiler Testing Center (FCTC). Also, all new acquisition of COBOL compilers by USACSC will be in accordance with the American National Standard Programing Language COBOL Standard, ANSI X3.23-1974 or subsequent to that release.

ACKNOWLEDGMENT. The following is an extract from the COBOL Journal of Development, a product of the CODASYL COBOL Programing Language Committee.

> 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 CODASYL Programing Language Committee as to the accuracy and functioning of the programing system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

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 programing manuals or similar publications.

> FLOW-MATIC (trademark of the Sperry Rand Corporation), Programing for the UNIVAC (R) I and II. Data Automation Systems copyrighted 1958, 1959 by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM: FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell. MetaCOBOL Facility copyrighted 1979 by Applied Data Research, Inc.

Supersedes CSCM 18-1-1, 1 Feb 79, and all changes.

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A

DTIC COPY INSPECTED 2

CONTENTS


| | Paragraph | Page |
|---|---|---|
| Chapter 1 - USACSC Programing Procedures | | |
| General | 1.1 | 1-1 |
| Introduction | 1.2 | 1-1 |
| Chapter 1 | 1.2.1 | 1-1 |
| Chapter 2 | 1.2.2 | 1-1 |
| Chapter 3 | 1.2.3 | 1-2 |
| Chapter 4 | 1.2.4 | 1-2 |
| Objectives | 1.3 | 1-2 |
| SPEC | 1.4 | 1-2 |
| SPEC Procedures | 1.4.1 | 1-2 |
| Exception to Use of SPEC | 1.4.2 | 1-2 |
| Exception to Use of ANSI COBOL | 1.4.3 | 1-3 |
| Authority to Grant an Exception | 1.4.4 | 1-3 |
| FORTRAN | 1.5 | 1-3 |
| FORTRAN Procedure | 1.5.1 | 1-3 |
| Exception to the Use of FORTRAN | 1.5.2 | 1-3 |
| Authority to Grant an Exception | 1.5.3 | 1-4 |
| Changes to Manual | 1.6 | 1-4 |
| USACSC Programing Concepts | 1.7 | 1-4 |
| The Simplistic Approach | 1.7.1 | 1-4 |
| Self-documenting Programs | 1.7.1.1 | 1-5 |
| Machine Independence | 1.7.1.2 | 1-5 |
| Maintainability | 1.7.1.3 | 1-5 |
| Productivity | 1.7.1.4 | 1-5 |
| Logical Flow | 1.7.1.5 | 1-5 |
| Standard Construct | 1.7.1.6 | 1-5 |
| Efficiency | 1.7.1.7 | 1-5 |
| Program Design Criteria | 1.8 | 1-6 |
| Program Identification | 1.9 | 1-6 |
| Multiple Program Outputs | 1.10 | 1-6 |
| File Organization | 1.11 | 1-7 |
| Sequential File Organization | 1.11.1 | 1-7 |
| Indexed File Organization | 1.11.2 | 1-7 |
| Random File Organization | 1.11.3 | 1-8 |
| Elements of File Design | 1.11.4 | 1-8 |
| File Design Considerations | 1.11.5 | 1-8 |
| Interactive Factors | 1.11.5.1 | 1-8 |
| Hit Ratio | 1.11.5.2 | 1-9 |
| Misleading Rules of File Design | 1.11.5.3 | 1-9 |
| Input Media | 1.11.6 | 1-10 |
| Console | 1.11.6.1 | 1-10 |

| | Paragraph | Page |
|---|---|---|
| Chapter 1 - USACSC Programing Procedures (Continued) | | |
| Tape | 1.11.6.2 | 1-10 |
| Card | 1.11.6.3 | 1-10 |
| Direct Access Storage Devices | 1.11.6.4 | 1-10 |
| Output Media | 1.11.7 | 1-10 |
| Printer | 1.11.7.1 | 1-11 |
| Punch | 1.11.7.2 | 1-12 |
| Tapes and Direct Access Storage Devices | 1.11.7.3 | 1-12 |
| Program to Operator Messages | 1.11.8 | 1-12 |
| Format | 1.11.8.1 | 1-13 |
| Program-ID | 1.11.8.2 | 1-13 |
| Message Number | 1.11.8.3 | 1-13 |
| Type of Message | 1.11.8.4 | 1-13 |
| Halts | 1.11.8.5 | 1-13 |
| Recovery Guidance | 1.11.9 | 1-14 |
| Error Condition Options | 1.11.10 | 1-14 |
| Console Switches | 1.11.11 | 1-14 |
| Utility Programs and Subroutines | 1.11.12 | 1-14 |
| Chapter 2 - USACSC Standard Portable Expanded COBOL, ANSI COBOL Subset | | |
| ANSI COBOL | 2.1 | 2-1 |
| Introduction | 2.2 | 2-1 |
| Purpose of USACSC Standard Portable Expanded COBOL, ANSI COBOL Subset Specifications | 2.2.1 | 2-1 |
| Language Conventions | 2.3 | 2-2 |
| Glossary | 2.3.1 | 2-2 |
| Language Structure | 2.3.2 | 2-2 |
| Separators | 2.3.2.1 | 2-2 |
| Punctuation Characters | 2.3.2.2 | 2-2 |
| Quotation Marks | 2.3.2.3 | 2-2 |
| Character-String | 2.3.2.4 | 2-2 |
| Delimiters | 2.3.2.5 | 2-2 |
| Words | 2.3.3 | 2-2 |
| Definition of a Word | 2.3.3.1 | 2-2 |
| Types of Words | 2.3.3.2 | 2-2 |
| Concepts of Data Reference | 2.3.4 | 2-6 |
| Logical Record and File Concept | 2.3.4.1 | 2-6 |
| Concept of COBOL Levels | 2.3.4.2 | 2-6 |
| Level-Numbers | 2.3.4.3 | 2-7 |
| Concepts of Classes of Data | 2.3.4.4 | 2-7 |
| Algebraic Signs | 2.3.4.5 | 2-8 |

| | Paragraph | Page |
|---|---|---|
| Chapter 2 - USACSC Standard Portable Expanded COBOL, ANSI COBOL Subset (Continued) | | |
| Qualification of Name | 2.3.4.6 | 2-8 |
| Subscripting | 2.3.4.7 | 2-9 |
| Indexing | 2.3.4.8 | 2-9 |
| USACSC COBOL Reference Format | 2.3.5 | 2-10 |
| General Description of Reference Format | 2.3.5.1 | 2-10 |
| Reference Format Representation | 2.3.5.2 | 2-10 |
| Data Division Entries | 2.3.5.3 | 2-12 |
| Continuation | 2.3.5.4 | 2-13 |
| Blank Lines | 2.3.5.5 | 2-14 |
| Comment Lines | 2.3.5.6 | 2-14 |
| USACSC Standard Coding Conventions | 2.3.6 | 2-14 |
| Coding | 2.3.6.1 | 2-14 |
| USACSC COBOL Specifications | 2.4 | 2-16 |
| Format Rules and Notes | 2.4.1 | 2-16 |
| Language Element | 2.4.1.1 | 2-16 |
| Function | 2.4.1.2 | 2-16 |
| Format | 2.4.1.3 | 2-16 |
| Syntax Rules | 2.4.1.4 | 2-16 |
| General Rules | 2.4.1.5 | 2-16 |
| USACSC Guidelines | 2.4.1.6 | 2-16 |
| Format Punctuation | 2.4.2 | 2-16 |
| General Description | 2.4.2.1 | 2-16 |
| Elements | 2.4.2.2 | 2-17 |
| Symbols and Notations Used in This Manual | 2.4.3 | 2-17 |
| General | 2.4.3.1 | 2-17 |
| Format Presentation | 2.4.3.2 | 2-17 |
| Default Option | 2.4.3.3 | 2-18 |
| Ellipsis | 2.4.3.4 | 2-19 |
| COBOL Program Structure | 2.4.4 | 2-20 |
| Divisions | 2.4.4.1 | 2-20 |
| Formats | 2.4.4.2 | 2-20 |
| Structure of the COBOL Program | 2.4.4.3 | 2-21 |
| IDENTIFICATION DIVISION | 2.4.5 | 2-23 |
| ELEMENTS | 2.4.5.1 | 2-23 |
| PROGRAM-ID PARAGRAPH | 2.4.5.2 | 2-24 |
| AUTHOR PARAGRAPH | 2.4.5.3 | 2-24 |
| INSTALLATION PARAGRAPH | 2.4.5.4 | 2-25 |

| | Paragraph | Page |
|---|---|---|
| Chapter 2 - USACSC Standard Portable Expanded COBOL, ANSI COBOL Subset (Continued) | | |
| DATE-WRITTEN PARAGRAPH | 2.4.5.5 | 2-25 |
| DATE-COMPILED PARAGRAPH | 2.4.5.6 | 2-26 |
| SECURITY PARAGRAPH | 2.4.5.7 | 2-26 |
| REMARKS PARAGRAPH | 2.4.5.8 | 2-27 |
| ENVIRONMENT DIVISION | 2.4.6 | 2-27 |
| ELEMENTS | 2.4.6.1 | 2-27 |
| CONFIGURATION SECTION | 2.4.6.2 | 2-28 |
| SOURCE-COMPUTER PARAGRAPH | 2.4.6.3 | 2-28 |
| OBJECT-COMPUTER PARAGRAPH | 2.4.6.4 | 2-29 |
| SPECIAL-NAMES PARAGRAPH | 2.4.6.5 | 2-30 |
| INPUT-OUTPUT SECTION | 2.4.6.6 | 2-32 |
| FILE-CONTROL PARAGRAPH | 2.4.6.7 | 2-33 |
| SELECT CLAUSE | 2.4.6.8 | 2-34 |
| ASSIGN CLAUSE | 2.4.6.9 | 2-35 |
| RESERVE CLAUSE | 2.4.6.10 | 2-47 |
| ACCESS CLAUSE | 2.4.6.11 | 2-47 |
| RELATIVE KEY CLAUSE | 2.4.6.12 | 2-48 |
| RECORD KEY CLAUSE | 2.4.6.13 | 2-50 |
| ALTERNATE RECORD KEY CLAUSE | 2.4.6.14 | 2-51 |
| FILE STATUS CLAUSE | 2.4.6.15 | 2-51 |
| I-O-CONTROL PARAGRAPH | 2.4.6.16 | 2-52 |
| RERUN CLAUSE | 2.4.6.17 | 2-55 |
| SAME CLAUSE | 2.4.6.18 | 2-56 |
| MULTIPLE FILE TAPE CLAUSE | 2.4.6.19 | 2-57 |
| DATA DIVISION | 2.4.7 | 2-58 |
| ELEMENTS | 2.4.7.1 | 2-58 |
| FILE SECTION | 2.4.7.2 | 2-60 |
| WORKING-STORAGE SECTION | 2.4.7.3 | 2-62 |
| LINKAGE SECTION | 2.4.7.4 | 2-66 |
| FILE DESCRIPTION (FD) AND SORT-FILE (SD) DESCRIPTION ENTRIES | 2.4.7.5 | 2-68 |
| LABEL RECORDS CLAUSE | 2.4.7.6 | 2-69 |
| RECORD CONTAINS CLAUSE | 2.4.7.7 | 2-70 |
| VALUE OF CLAUSE | 2.4.7.8 | 2-70 |
| BLOCK CONTAINS CLAUSE | 2.4.7.9 | 2-71 |
| CODE-SET CLAUSE | 2.4.7.10 | 2-73 |
| DATA RECORD CLAUSE | 2.4.7.11 | 2-74 |
| RECORD DESCRIPTION CLAUSE | 2.4.7.12 | 2-74 |
| DATA DESCRIPTION CLAUSE | 2.4.7.13 | 2-75 |
| DATA-NAME OR FILLER CLAUSE | 2.4.7.14 | 2-77 |
| REDEFINES CLAUSE | 2.4.7.15 | 2-78 |
| SIGN CLAUSE | 2.4.7.16 | 2-79 |
| OCCURS CLAUSE | 2.4.7.17 | 2-81 |
| PICTURE CLAUSE | 2.4.7.18 | 2-85 |
| USAGE CLAUSE | 2.4.7.19 | 2-92 |
| VALUE CLAUSE | 2.4.7.20 | 2-93 |
| JUSTIFIED CLAUSE | 2.4.7.21 | 2-95 |
| SYNCHRONIZED CLAUSE | 2.4.7.22 | 2-96 |
| BLANK WHEN ZERO CLAUSE | 2.4.7.23 | 2-98 |

| | Paragraph | Page |
|---|---|---|
| Chapter 2 - USACSC Standard Portable Expanded COBOL, ANSI COBOL Subset (Continued) | | |
| Procedure Division | 2.4.8 | 2-99 |
| General Description | 2.4.8.1 | 2-99 |
| Structure | 2.4.8.2 | 2-99 |
| General Rules | 2.4.8.3 | 2-100 |
| STATEMENTS | 2.4.9 | 2-121 |
| ACCEPT STATEMENT | 2.4.9.1 | 2-121 |
| ADD STATEMENT | 2.4.9.2 | 2-123 |
| ALTER STATEMENT | 2.4.9.3 | 2-126 |
| CALL STATEMENT | 2.4.9.4 | 2-127 |
| CANCEL STATEMENT | 2.4.9.5 | 2-129 |
| CASE STATEMENT | 2.4.9.6 | 2-130 |
| CLOSE STATEMENT | 2.4.9.7 | 2-131 |
| COMPUTE STATEMENT | 2.4.9.8 | 2-138 |
| COPY STATEMENT | 2.4.9.9 | 2-142 |
| USE FOR DEBUGGING STATEMENT | 2.4.9.10 | 2-145 |
| DEBUG-ITEM Special Register | 2.4.9.11 | 2-147 |
| DELETE STATEMENT | 2.4.9.12 | 2-149 |
| DISPLAY STATEMENT | 2.4.9.13 | 2-150 |
| DIVIDE STATEMENT | 2.4.9.14 | 2-152 |
| DO STATEMENT | 2.4.9.15 | 2-155 |
| DO UNTIL STATEMENT | 2.4.9.16 | 2-156 |
| DO WHILE STATEMENT | 2.4.9.17 | 2-157 |
| ENTER STATEMENT | 2.4.9.18 | 2-158 |
| EXIT STATEMENT | 2.4.9.19 | 2-159 |
| GO TO STATEMENT | 2.4.9.20 | 2-160 |
| IF STATEMENT | 2.4.9.21 | 2-161 |
| INSPECT STATEMENT | 2.4.9.22 | 2-166 |
| MOVE STATEMENT | 2.4.9.23 | 2-174 |
| MULTIPLY STATEMENT | 2.4.9.24 | 2-179 |
| OPEN STATEMENT | 2.4.9.25 | 2-181 |
| PERFORM STATEMENT | 2.4.9.26 | 2-185 |
| READ STATEMENT | 2.4.9.27 | 2-195 |
| RELEASE STATEMENT | 2.4.9.28 | 2-198 |
| RETURN STATEMENT | 2.4.9.29 | 2-199 |
| REWRITE STATEMENT | 2.4.9.30 | 2-200 |
| SEARCH STATEMENT | 2.4.9.31 | 2-202 |
| SET STATEMENT | 2.4.9.32 | 2-203 |
| SORT STATEMENT | 2.4.9.33 | 2-204 |
| START STATEMENT | 2.4.9.34 | 2-213 |
| STOP STATEMENT | 2.4.9.35 | 2-214 |
| SUBTRACT STATEMENT | 2.4.9.36 | 2-216 |
| USE STATEMENT | 2.4.9.37 | 2-219 |
| WRITE STATEMENT | 2.4.9.38 | 2-221 |
| Special Features | 2.5 | 2-226 |
| Structured Programing Statements - MetaCOBOL Macro Facility | 2.5.1 | 2-226 |
| DO STATEMENT | 2.5.1.1 | 2-227 |
| DO WHILE STATEMENT | 2.5.1.2 | 2-228 |

| | Paragraph | Page |
|---|---|---|
| Chapter 2 - USACSC Standard Portable Expanded COBOL, ANSI COBOL Subset (Continued) | | |
| IF STATEMENT | 2.5.1.3 | 2-230 |
| DO UNTIL STATEMENT | 2.5.1.4 | 2-232 |
| CASE STATEMENT | 2.5.1.5 | 2-233 |
| COBOL Segmentation Facility | 2.5.2 | 2-235 |
| Organization of Segmentation Facility | 2.5.2.1 | 2-235 |
| Segment Classification | 2.5.2.2 | 2-236 |
| Segmentation Control | 2.5.2.3 | 2-236 |
| Structure of Program Segments | 2.5.2.4 | 2-236 |
| Restrictions on PERFORM Statement | 2.5.2.5 | 2-237 |
| Example of Segmentation | 2.5.2.6 | 2-238 |
| USACSC Guidelines for Segmentation | 2.5.2.7 | 2-239 |
| Sort Feature | 2.5.3 | 2-240 |
| Introduction | 2.5.3.1 | 2-240 |
| Environment Division Sort Feature | 2.5.3.2 | 2-241 |
| Data Division Sort Feature | 2.5.3.3 | 2-241 |
| Procedure Division Sort Feature | 2.5.3.4 | 2-242 |
| Table Handling Feature | 2.5.4 | 2-242 |
| Introduction | 2.5.4.1 | 2-242 |
| Subscripting | 2.5.4.2 | 2-243 |
| OCCURS STATEMENT | 2.5.4.3 | 2-247 |
| Indexing | 2.5.4.4 | 2-249 |
| Procedure Division Considerations for Table Handling | 2.5.4.5 | 2-251 |
| Source Program Library Facility | 2.5.5 | 2-261 |
| Introduction to Copy Library Facility | 2.5.5.1 | 2-261 |
| COPY STATEMENT | 2.5.5.2 | 2-262 |
| Debugging Aids | 2.5.6 | 2-269 |
| Introduction to Debugging Aids | 2.5.6.1 | 2-269 |
| DOS COBOL Program Debugging Aids | 2.5.6.2 | 2-271 |
| Common Causes of Errors | 2.5.6.3 | 2-272 |
| Link Edit Map | 2.5.6.4 | 2-273 |
| Object Storage Layout | 2.5.6.5 | 2-274 |
| System Action Under Cancel | 2.5.6.6 | 2-285 |
| Wait States | 2.5.6.7 | 2-296 |
| Commonly Encountered User Errors | 2.5.6.8 | 2-297 |
| DOS Core Dump Tracing | 2.5.6.9 | 2-299 |
| Interpreting Output | 2.5.6.10 | 2-309 |
| OS COBOL Program Debugging Aids | 2.5.6.11 | 2-326 |
| COMPLETION CODE - ØØ1 | 2.5.6.12 | 2-326 |
| COMPLETION CODE - Ø13 | 2.5.6.13 | 2-328 |
| COMPLETION CODE - Ø31 | 2.5.6.14 | 2-328 |
| COMPLETION CODE - Ø3B | 2.5.6.15 | 2-330 |
| COMPLETION CODE - Ø3D | 2.5.6.16 | 2-331 |
| OCx COMPLETION CODE NOTE | 2.5.6.17 | 2-331 |

| | Paragraph | Page |
|---|---|---|
| Chapter 2 - USACSC Standard Portable Expanded COBOL, ANSI COBOL Subset (Continued) | | |
| COMPLETION CODE - 0C1 | 2.5.6.18 | 2-331 |
| COMPLETION CODE - 0C5 | 2.5.6.19 | 2-332 |
| COMPLETION CODE - 0C7 | 2.5.6.20 | 2-333 |
| COMPLETION CODE - 237 | 2.5.6.21 | 2-334 |
| COMPLETION CODE - 637 | 2.5.6.22 | 2-336 |
| COMPLETION CODE - 804 | 2.5.6.23 | 2-338 |
| COMPLETION CODE - 806 | 2.5.6.24 | 2-338 |
| COMPLETION CODE - 813 | 2.5.6.25 | 2-339 |
| COMPLETION CODE - D37 | 2.5.6.26 | 2-340 |
| COMPLETION CODE - E37 | 2.5.6.27 | 2-341 |
| Control Block Pointers | 2.5.6.28 | 2-342 |
| OS/MVT Core Dump | 2.5.6.29 | 2-348 |
| OS Data Exceptions, Recognition and Error Recovery | 2.5.6.30 | 2-356 |
| Register and Save Area | 2.5.6.31 | 2-357 |
| OC7 (Data Check) Debugging Exercise | 2.5.6.32 | 2-358 |
| Debugging of COBOL Segmented Programs Under OS/MFT | 2.5.6.33 | 2-368 |
| USACSC COBOL Program Design Techniques | 2.6 | 2-372 |
| Procedure Division Design | 2.6.1 | 2-372 |
| Standard Logic Constructs | 2.6.2 | 2-373 |
| COBOL Program Structure Techniques | 2.7 | 2-381 |
| Data Format Considerations | 2.7.1 | 2-383 |
| Data Item Considerations | 2.7.2 | 2-386 |
| Procedure Division Techniques | 2.7.3 | 2-396 |
| Paragraph Naming | 2.7.3.1 | 2-396 |
| File Processing | 2.7.3.2 | 2-396 |
| Conditional Statements | 2.7.3.3 | 2-397 |
| Arithmetic Operations | 2.7.3.4 | 2-402 |
| Branching Statements | 2.7.3.5 | 2-405 |
| Data Manipulation | 2.7.3.6 | 2-408 |
| Table Handling Techniques | 2.7.4 | 2-408 |
| Table Construction and Referencing | 2.7.4.1 | 2-408 |
| Transfer of Control | 2.7.5 | 2-414 |
| Overlay Structures | 2.7.5.1 | 2-414 |
| Subprogram Linkage | 2.7.5.2 | 2-415 |
| Subprogram Technique | 2.7.5.3 | 2-416 |
| Source Language System (SLS)/Program Language Update Service (PLUS) | 2.7.6 | 2-418 |
| Source Library Maintenance | 2.7.6.1 | 2-418 |
| Catalogued Programs | 2.7.6.2 | 2-418 |
| Single Source Library System | 2.8 | 2-418 |
| Objective | 2.8.1 | 2-418 |
| Procedures | 2.8.2 | 2-419 |
| Coding | 2.8.3 | 2-419 |
| Single Source System | 2.8.4 | 2-419 |
| Implementing Instructions | 2.8.5 | 2-419 |

| | Paragraph | Page |
|---|---|---|
| Chapter 2 - USACSC Standard Portable Expanded COBOL, ANSI COBOL Subset (Continued) | | |
| OS/DOS Compatibility | 2.9 | 2-420 |
| Program Techniques | 2.9.1 | 2-420 |
| Input/Storage Areas | 2.9.2 | 2-420 |
| Input Buffers | 2.9.2.1 | 2-420 |
| Address Pointer | 2.9.2.2 | 2-420 |
| STOP RUN STATEMENT | 2.9.3 | 2-420 |
| Data Formats | 2.9.4 | 2-421 |
| Record Identifier | 2.9.5 | 2-421 |
| Program Switches | 2.9.6 | 2-421 |
| PICTURE CLAUSE | 2.9.7 | 2-422 |
| APPLY CLAUSE | 2.9.8 | 2-422 |
| Invalid Key Option | 2.9.9 | 2-422 |
| Syntax Errors | 2.9.10 | 2-422 |
| Chapter 3 - USACSC Structured Programing Technology | | |
| Introduction | 3.1 | 3-1 |
| General | 3.1.1 | 3-1 |
| Purpose | 3.1.2 | 3-1 |
| Definitions | 3.1.3 | 3-1 |
| Backup Programer | 3.1.3.1 | 3-1 |
| Chief Programer | 3.1.3.2 | 3-2 |
| Data Flow Graph | 3.1.3.3 | 3-2 |
| IPO (Input, Process, Output) Chart | 3.1.3.4 | 3-2 |
| Librarian | 3.1.3.5 | 3-2 |
| Program Design Language (PDL) or PSEUDO-CODE | 3.1.3.6 | 3-2 |
| Programing Support Library (PSL) | 3.1.3.7 | 3-2 |
| Structure Chart | 3.1.3.8 | 3-2 |
| Structured Program | 3.1.3.9 | 3-3 |
| Structured Programing (SP) or Structured Coding (SC) | 3.1.3.10 | 3-3 |
| Structured Source Code Listing | 3.1.3.11 | 3-3 |
| Structured Testing | 3.1.3.12 | 3-4 |
| Structured Walkthrough | 3.1.3.13 | 3-4 |
| Stub | 3.1.3.14 | 3-4 |
| Team Operation or Chief Programer Team | 3.1.3.15 | 3-4 |
| Top Down Development | 3.1.3.16 | 3-4 |
| Top Down Program (TDP) | 3.1.3.17 | 3-5 |
| Top Down Structured Programing (TDSP) | 3.1.3.18 | 3-5 |
| Concepts of Top Down Structured Programing (TDSP) | 3.1.4 | 3-5 |
| General | 3.1.4.1 | 3-5 |
| Figures | 3.1.4.2 | 3-6 |
| Program Design Language (PDL) or PSEUDO-CODE | 3.1.4.3 | 3-6 |
| Programing Support Library (PSL) | 3.1.4.4 | 3-11 |
| Structured Walkthroughs | 3.1.4.5 | 3-13 |
| Chief Programer Team (CPT) | 3.1.4.6 | 3-14 |
| Nine Step Module Management Process | 3.1.4.7 | 3-15 |

| | Paragraph | Page |
|---|---|---|
| Chapter 3 - USACSC Structured Programing Technology (Continued) | | |
| General Standards and Guidelines | 3.1.5 | 3-16 |
| Standards | 3.1.5.1 | 3-16 |
| Guidelines | 3.1.5.2 | 3-17 |
| USACSC SPEC COBOL Language Standards and Guidelines | 3.1.6 | 3-17 |
| Include Capability | 3.1.7 | 3-17 |
| Additional Recommended Coding Conventions | 3.1.7.1 | 3-18 |
| Chapter 4 - FORTRAN Programing Procedures | | |
| Introduction | 4.1 | 4-1 |
| Design Considerations | 4.2 | 4-1 |
| Modularity | 4.2.1 | 4-1 |
| Library Functions | 4.2.2 | 4-1 |
| Input/Output Functions | 4.2.3 | 4-1 |
| Real And Integer Data | 4.2.4 | 4-1 |
| Program Structure | 4.3 | 4-1 |
| Source Card Coding | 4.3.1 | 4-1 |
| Comment Cards | 4.3.1.1 | 4-2 |
| Sequence Numbers | 4.3.1.2 | 4-2 |
| Statement Labeling | 4.3.1.3 | 4-2 |
| Statement Ordering | 4.3.1.4 | 4-3 |
| Symbolic Names | 4.3.1.5 | 4-3 |
| FORTRAN Character Set | 4.3.2 | 4-3 |
| Operators Used in FORTRAN Programs | 4.3.3 | 4-4 |
| Arithmetic Expressions | 4.3.3.1 | 4-4 |
| Relation Operators | 4.3.3.2 | 4-4 |
| Logical Operators | 4.3.3.3 | 4-4 |
| Additional Information | 4.3.3.4 | 4-5 |
| Arrays | 4.4 | 4-5 |
| Variable Names | 4.5 | 4-5 |
| Program Comments | 4.6 | 4-5 |
| Meaningful Comments | 4.6.1 | 4-5 |
| Identification of Program in a Comment | 4.6.2 | 4-5 |
| Program Modification | 4.6.3 | 4-6 |
| Program Comments for Subroutines | 4.6.4 | 4-6 |
| Distribute Comment | 4.6.5 | 4-6 |
| Descriptive Comments | 4.6.6 | 4-6 |
| Conspicuous Printing Style for Comment | 4.6.7 | 4-6 |
| Recovery Procedures In Comments | 4.6.8 | 4-6 |
| Check And Desk Checking | 4.7 | 4-7 |
| Checkout Method | 4.7.1 | 4-7 |
| Desk Checking | 4.7.2 | 4-7 |
| Program Logic Checklist | 4.7.3 | 4-7 |
| Statement Number | 4.7.3.1 | 4-7 |
| Verify Statement Number | 4.7.3.2 | 4-7 |
| Assure Parentheses Balance | 4.7.3.3 | 4-7 |
| Subscripted Variables | 4.7.3.4 | 4-7 |
| Check For DO-loop | 4.7.3.5 | 4-7 |

| | Paragraph | Page |
|---|---|---|
| Assure Statements Are Present | 4.7.3.6 | 4-7 |
| Check All Hollerith Fields | 4.7.3.7 | 4-7 |
| CALL Statement | 4.7.3.8 | 4-7 |
| Arguments | 4.8 | 4-7 |
| Grouping of Arguments | 4.8.1 | 4-8 |
| Error Code | 4.8.2 | 4-8 |
| Non-integer Variable | 4.9 | 4-8 |
| Array Naming Convention | 4.9.1 | 4-8 |
| Arguments In Call Statements | 4.9.2 | 4-8 |
| Data Variable Assignment | 4.9.3 | 4-8 |
| Whole Numbers | 4.9.4 | 4-8 |
| Input/Output Devices | 4.9.5 | 4-8 |
| Constant Count Indices | 4.9.6 | 4-8 |

PAGE

ATTACHMENTS
ATTACHMENT 1 - GLOSSARY A-1
ATTACHMENT 2 - RESERVED WORDS B-1

INDEX Index-1

FIGURES
1-1 1-8
1-2 1-12
1-3 1-13
1-4 1-13
2-1 2-4
2-2 2-8
2-3 2-15
2-4 2-21
2-5 2-44
2-6 2-45
2-7 2-63
2-8 2-89
2-9 2-89
2-10 2-90
2-11 2-91
2-12 2-91
2-13 2-101
2-14 2-103
2-15 2-104
2-16 2-105
2-17 2-108
2-18 2-111
2-19 2-114
2-20 2-117
2-21 2-118
2-22 2-120
2-23 2-132
2-24 2-148
2-25 2-165
2-26 2-177
2-27 2-189
2-28 2-191
2-29 2-192
2-30 2-209
2-31 2-238
2-32 2-245
2-33 2-255
2-34 2-258
2-35 2-262
2-36 2-265
2-37 2-275
2-38 2-279
2-39 2-281
2-40 2-282
2-41 2-294
2-42 2-295
2-43 2-295

| FIGURES | PAGE |
|---|---|
| 2-44 | 2-302 |
| 2-45 | 2-312 |
| 2-46 | 2-315 |
| 2-47 | 2-319 |
| 2-48 | 2-319 |
| 2-49 | 2-321 |
| 2-50 | 2-325 |
| 2-51 | 2-353 |
| 2-52 | 2-362 |
| 2-53 | 2-363 |
| 2-54 | 2-364 |
| 2-55 | 2-365 |
| 2-56 | 2-366 |
| 2-57 | 2-367 |
| 2-58 | 2-370 |
| 2-59 | 2-371 |
| 2-60 | 2-374 |
| 2-61 | 2-375 |
| 2-62 | 2-378 |
| 2-63 | 2-379 |
| 2-64 | 2-381 |
| 2-65 | 2-414 |
| 2-66 | 2-416 |
| 3-1 | 3-8 |
| 3-2 | 3-9 |
| 3-3 | 3-9 |

# CHAPTER 1

## USACSC PROGRAMING PROCEDURES

1.1 GENERAL. This manual prescribes procedures to be used by programers in developing and maintaining multicommand or Command-unique US Army Computer Systems Command (USACSC) managed systems. It will be used as the primary source of information for USACSC programers.

1.2 INTRODUCTION.

1.2.1 CHAPTER 1. This chapter is general in scope and is not related to a specific language. The material contained in Chapter 1 is considered mandatory.

1.2.2 CHAPTER 2. This chapter deals with the Standard Portable Expanded COBOL (SPEC) specifications and procedures and is considered mandatory with the following specific guidance:

- SPEC is a portable dialect of American National Standards Institute (ANSI) 1974 COBOL.

- If an element of ANSI COBOL has not been described in the Programing Procedures Manual, it will not be used.

- All new programs will follow the standards set forth in this chapter.

- Existing programs or systems will normally be changed to conform to these standards only when other changes are being made to the programs or systems, where the standards changes may be made in conjunction. Under proper conditions, standards changes may be made alone, without accompanying logic changes, but under no conditions will any changes be made in any programs or systems without formal submission of a Systems Change Request (SCR) through the normal procedures as spelled out in AR 18-1, and USACSC Regulations 18-1 and 18-21.

- The subsections on COBOL Program Design Techniques and USACSC COBOL Programing Techniques in this chapter are not mandatory. They represent the preferred approaches and are to be used unless clearly not applicable to the problem at hand.

1.2.2 CHAPTER 2. (Cont.)

- Throughout the discussion of the SPEC language elements, USACSC guidelines are given. When the operatives "should", "ideally", "it is recommended", etc., are employed, the guidelines identify "the preferred method approach" rather than mandatory rules.

- If a prohibited feature is needed due to systems requirements, a waiver for that specific contravention of command standards must be sought through the procedures as outlined in TB 18-115, Army Information Processing Standards (AIPS) Program.

1.2.3 CHAPTER 3. This chapter addresses STRUCTURED PROGRAMING.

1.2.4 CHAPTER 4. This chapter addresses FORTRAN PROGRAMING PROCEDURES. Subsequent sections will deal with other languages as required.

1.3 OBJECTIVES. Standard programing languages and uniform software conventions are adopted to achieve the following:

- Centralized preparation of programs.

- Capability of transferring programs from one operating system to another or from one hardware manufacturer to another with the minimum amount of programer effort.

- Simplification in programing by use of a language oriented to subject matter terms.

- Greater sharing of computer programs.

1.4 SPEC. ANSI COBOL is the DA standard computer language for use in business-type information and data systems. SPEC is a USACSC defined portable dialect of ANSI COBOL.

1.4.1 SPEC PROCEDURES. Developers responsible for business-type systems design and/or programing will insure that the following procedures are followed:

- Program all new systems in SPEC, except as noted below.

- Program major revisions to existing systems in SPEC when the use of SPEC will not adversely affect system capability.

- Reprogram existing non-SPEC programs to SPEC as time and funds permit.

1.4.2 EXCEPTION TO USE OF SPEC. An exception to the use of SPEC does not necessarily imply that an exception to the use of COBOL will be granted. An exception to the use of SPEC may be authorized under the following conditions:

1.4.2 EXCEPTION TO USE OF SPEC. (Cont.)

- A SPEC compiler/translator does not exist for the target hardware.

- A SPEC compiler/translator exists but the target hardware configuration cannot adequately accommodate the application if written in SPEC.

1.4.3 EXCEPTION TO USE OF ANSI COBOL. An exception to the use of ANSI COBOL may be authorized under the following conditions:

- A COBOL compiler does not exist for the target hardware.

- A COBOL compiler exists but the configuration in use cannot adequately accommodate the compiler because of memory or equipment limitations.

- A COBOL compiler exists but certain required processing conditions cannot be handled, examples being random access processing, translation of an ASC II encoded tape, etc., such problems being solvable by called ALC subroutines. The requirements for such modules should be addressed to the Executive Software Division of the Executive Software Systems Directorate for consideration for inclusion in Command executive software before a waiver is requested for a system-unique ALC module.

- The cost of compiling ANSI COBOL programs has been properly documented and determined to be prohibitive.

1.4.4 AUTHORITY TO GRANT AN EXCEPTION. The Commander, US Army Computer Systems Command is authorized to grant an exception to the use of ANSI COBOL on all multicommand or Command-unique systems that operate on ADPE acquired by the Department of the Army to support standard USACSC-managed multicommand systems. The request will be submitted in accordance with the provisions of TB 18-115, Army Information Processing Standards (AIPS) Program.

1.5 FORTRAN. American National Standard FORTRAN (ANSI X3.9-1966) and basic FORTRAN (ANSI X3.10-1966) are designated the Army Standard Programing Language (ASPL) for scientific and engineering applications. Use of language elements provided in individual FORTRAN compilers but not defined in the FORTRAN standards above must be approved by HQDA.

1.5.1 FORTRAN PROCEDURE. Developers responsible for scientific and engineering type systems design and/or programing will program new systems and major revisions to existing systems in FORTRAN.

1.5.2 EXCEPTION TO THE USE OF FORTRAN. An exception to the use of FORTRAN in programs may be authorized under the following circumstances:

- A FORTRAN compiler does not exist for the model of computer.

- A FORTRAN compiler exists for the configuration but lacks the power and flexibility necessary to operate efficiently in a given system.

1.5.2 EXCEPTION TO THE USE OF FORTRAN. (Cont.)

- A FORTRAN compiler exists for the configuration but lacks required capabilities (i.e., floating point) and does not permit exit to another language.

1.5.3 AUTHORITY TO GRANT AN EXCEPTION. The Commander, US Army Computer Systems Command is authorized to grant an exception to the use of FORTRAN on all multicommand or Command-unique systems that operate on ADPE acquired by the Department of the Army to support standard USACSC-managed multicommand systems. The request will be submitted in accordance with the provisions of TB 18-115, Army Information Processing Standards (AIPS) Program.

1.6 CHANGES TO MANUAL. Recommended changes to the manual are to be forwarded to Commander, US Army Computer Systems Command, ATTN: ACSC-TES, Fort Belvoir, VA 22060. Recommendations will be submitted on DA Form 2028. Incomplete DA Forms 2028 will be returned to the sender for further clarification before any action is taken.

1.7 USACSC PROGRAMING CONCEPTS. The following are general USACSC programing concepts:

- Easily transferable programs and programers that are machine independent and programs that are easily transferable or require a minimum translation and management effort to be fully operational on a variety of hardware.

- Easily maintained programs that are self-documenting with meaningful user defined words, concise and descriptive comments, and a clear "meaningful" structure of sentences and paragraphs.

- Easily debugged programs through modular, building block structure and avoidance of overly complex language elements.

- Efficient use of programing language to reduce processing time, simplify the operator's job, and make more effective use of peripheral devices and main storage.

1.7.1 THE SIMPLISTIC APPROACH. The best programing technique is the simplest one. The program which is kept simple is easier to understand and follow. The command programing objectives below are supported by the simplistic approach:

- Self-documenting Programs
- Machine Independence
- Maintainability
- Productivity

1.7.1 THE SIMPLISTIC APPROACH. (Cont.)

- Standard Construct
- Logical Flow
- Efficiency

1.7.1.1 Self-documenting Programs. These aid in developing, maintaining and reusing programs. The meaning and usage of data items is the most important, but most neglected, program documentation. Reasons for redefining data items as alphanumeric or numeric and class selection can be easily identified by providing a brief comment to the data item description. Other recommendations for developing self-documenting programs are covered in this manual. Suggestions include meaningful labels, data and logic organization.

1.7.1.2 Machine Independence. Independence from machine implies transferability and portability. Transferability is the ability to move a system from one vendor to another. Portability is the ability to move a system from one location to another in a business sense. Thus a payroll system may be used by more than one office or division on dissimilar equipment.

1.7.1.3 Maintainability. This implies a quick, accurate understanding of a program's functions and logic by the maintenance programer who may not be familiar with the program being maintained. This manual offers guidelines on location coding for labels, prefixing of data names, meaningful data, and program layout as well as simple, well-organized program logic. All of these will help a programer write a more readable, maintainable program.

1.7.1.4 Productivity. This is the accurate measure of a program's success - the long-range goal which all these other operational goals are designed to achieve. Meeting these goals will result in a successful, productive program.

1.7.1.5 Logical Flow. This increases the simplicity of the code. Subtle complexity is often introduced into the branching structure of a program. If a programer gets into difficulty, the temptation is very great to patch around a problem area. Many of the guidelines and standard constructs will help the programer to avoid these pitfalls.

1.7.1.6 Standard Construct. This is a standardized grouping of various language facilities designed to perform a particular function. Some of the constructs covered in this manual include the loop-controlling and flag use. Other techniques covered which improve logical flow include mainline, subroutines, and proper use of program control structures.

1.7.1.7 Efficiency. This is the aspect which programers are first concerned about. Core-usage and run-time are still important, but not the only objectives of the programer. They must be considered in conjunction with the previously mentioned objectives. In this manner the techniques section offers suggestions for efficient use of core and machine time.

1.8 PROGRAM DESIGN CRITERIA. This paragraph cannot cover all the decisions the programer must face. It does, however, cover general procedures applicable to any programing language.

• As the programer is designing his program, he can ask himself several questions which match the effects of each potential usage of a language element against the programing objectives of the command.

IS the element INEFFECTIVE or a repetitious, unnecessary alternative?

IS the coding AMBIGUOUS to the programer, if not to the compiler?

DOES it produce INEFFICIENT OBJECT CODE?

IS the element unnecessarily COMPLEX causing debugging and maintenance problems?

DO IMPLEMENTATIONS of the element VARY from machine to machine significantly, thus reducing chances of transferability?

IS the coding TOO GENERAL at the cost of inexplicit documentation?

IS the coding FREQUENTLY MISINTERPRETED by programers?

IS the coding convenient at the cost of INEFFECTIVE OPERATIONS?

DOES the procedure result in a program with POOR ORGANIZATION?

DOES the procedure cause future program MAINTENANCE PROBLEMS?

• One must recognize that some of the objectives reflected by the above questions are mutually exclusive. The compliance to one of the objectives may force lack of compliance to other objectives. However, the techniques suggested in this chapter are those which are designed to meet the objectives in the best manner.

1.9 PROGRAM IDENTIFICATION. The program identification will be assigned as outlined in TB 18-103, Software Design and Development.

1.10 MULTIPLE PROGRAM OUTPUTS. Parameter or control cards will be used to control production of more than one product by a single program. For example, a program can be written which produces several reports from a single source of input. The parameter or control card will contain an indicator representing which report is required. Under no circumstances will the console operator be required to make such a determination. Programs which produce variable or optional reports will provide suitable methods for programatically selecting or cancelling a particular report without requiring the console operator to make such a determination.

## 1.11 FILE ORGANIZATION.

### 1.11.1 SEQUENTIAL FILE ORGANIZATION.

• A sequentially organized file has records arranged in a specified order, according to a key or entry sequence. Access is serial. Sequential file organization can be used with either serial or direct access storage devices. However, all records must be serially passed in file sequence until the desired one is located. Fast access to individual records is not possible. As a result, transactions to update files are batched and arranged in the same order as the master file. The master file is updated as discussed below.

• When sequentially organized files stored on magnetic tape are updated, a new tape is created and the old tape provides the backup. Sequentially organized files on disks can be updated by developing and storing the new file in a separate section of disk storage, or by writing the new records in the original locations. But this latter procedure creates a backup problem. One solution is to read the old file onto magnetic tape prior to the updating operation.

• If a large number of records within a file are normally accessed at a given time, the sequential access method can permit faster and more efficient processing than the direct access method while providing reasonably fast access to stored data when only a few records are to be accessed.

### 1.11.2 INDEXED FILE ORGANIZATION.

• A file whose organization is indexed is a mass storage file in which data records may be accessed by the value of a key. A record description may include one or more key data items, each of which is associated with an index. Each index provides a logical path to the data records according to the contents of a data item within each record which is the record key for that index.

• The data item named in the RECORD KEY clause of the file control entry for a file is the prime record key for that file. For purposes of inserting, updating and deleting records in a file, each record is identified solely by the value of its prime record key. This value must, therefore, be unique and must not be changed when updating the record.

• A data item named in the ALTERNATE RECORD KEY clause of the file control entry for a file is an alternate record key for that file. The value of an alternate record key may be non-unique if the DUPLICATES phrase is specified for it. These keys provide alternate access paths for retrieval of records from the file.

• In the sequential access mode, the sequence in which records are accessed is the ascending order of the record key values. The order of retrieval of records within a set of records having duplicate record key values is the order in which the records were written into the set.

• In the random access mode, the sequence in which records are accessed is controlled by the programer. The desired record is accessed by placing the value of its record key in a record key data item.

• In the dynamic access mode, the programer may change at will from sequential access to random access using appropriate forms of input-output statements.

1.11.3 RANDOM FILE ORGANIZATION.

- When files are organized on a random basis, successive records are not stored in sequential order nor even necessarily in adjacent storage. Each record is located at an address which is computed by a randomizing process.

- Rewriting an entire file when additions or deletions are made is not necessary. Transaction data need not be batched or presorted prior to processing.

- Random file organization is usually best suited for situations in which large files are to be handled, the number of transactions per time period is not large and very fast processing is desirable.

1.11.4 ELEMENTS OF FILE DESIGN. FIGURE 1-1 summarizes the discussion of file organization.

| | ACTIVITY | VOLATILITY | SPACE UTILIZATION | ACCESS TIME |
|---|---|---|---|---|
| SEQUENTIAL | HIGH | LOW | HIGH | SLOW |
| INDEX SEQ. SEQ HIGH DIRECT LOW | | MED | LOW | FAST |
| RANDOM | LOW | HIGH | LOW | FAST |

FIGURE 1-1

1.11.5 FILE DESIGN CONSIDERATIONS.

1.11.5.1 Interacting Factors. A number of interacting factors must be considered in designing a file. These file design considerations, which are listed below, are closely related and no decision should be made in isolation. The first four considerations are program independent. The considerations are listed in the order they might logically occur, not in order of significance. For example, a block size might be crucial on a small configuration, but trivial on a large one.

| | |
|---|---|
| * DATA CONTENT | Which data items make up a record; which records make up a file. |
| * SEQUENCE | Random or sequential; if sequential, what sequence. |
| * FORMAT | Fixed or variable length records; location of data items within the record. |
| * PROCESSING MODE | Sequential, indexed or random. |
| * SECURITY | |
| * RESTART POINTS | |
| * DEVICES AVAILABLE | |

1.11.5.1 Interacting Factors. (Cont.)

* BUFFERING

* BLOCK SIZE

1.11.5.2 Hit Ratio. The hit ratio, which is a mathematical function used to describe file activity, is widely used to determine file organization.

- Hit Ratio = $\frac{\text{number of records accessed}}{\text{number of records on file}}$

- Just as the file design factors listed before are not independent of each other, hit ratio cannot be considered by itself. For example, time and type of activity can both influence hit ratio. Does the input activity have a peak or heavy activity? Is there a season or cycle? Perhaps on a given day only 5% of customers on a file place an order but 60% of the customers may be billed. The user has two hit ratios to consider.

- There are other factors at work which can be mentioned in passing such as track hit ratio, I/O message ratio, file volatility and overflow characteristics.

- Therefore, the hit ratio is not a simple construct which stands alone. The file designer must consider many other factors that are interactive.

1.11.5.3 Misleading Rules of File Design.

- If hit ratio is low, update the file randomly, otherwise sequentially.

THE CHOICE here is apparently between randomly updating hit records in position and updating the entire file. But there are other factors to take into consideration. For example, random processing infers the entire file to be on-line to the CPU; further if fast restart recovery procedures are required, the dumping of the original file records demands further secondary storage. All of this required storage space may not be available.

ANOTHER SOLUTION to the sequential versus random dilemma may be to split the file into hit and non-hit (or infrequently hit) sections. In this case, processing sequentially the infrequently referenced data at less frequent intervals might be justifiable.

CONVERSELY, RANDOM PROCESSING might be used in a high hit ratio situation For example, if data cause a high degree of interaction among separate files, then one-shot processing might be desirable whereby the data updates one file sequentially and the remaining files randomly.

- If hit ratio is low, use indexed-sequential; else use sequential processing

IF THE FILE is highly volatile or has severe overflow characteristics, whatever the hit ratio, another organization would probably be chosen.

1.11.5.3 Misleading Rules of File Design. (Cont.)

- If the hit ratio is low, use disk; otherwise, magnetic tape.

THIS AXIOM is the most general of all and embodies confusion between file organization methods, processing methods and devices. File organization and processing were discussed above and therefore it remains to compare disk and tape for sequential processing.

DISKS are faster than most tapes when sequentially processing; thus a tight time requirement might favor disks. Additionally, sequentially processing disk might require fewer devices than a similar tape system.

EVEN SEQUENTIAL DISK FILES can have characteristics of direct access when required, if only by the "binary search" technique; this facility might save passes of the file. For example, amendments to record keys can require that the file be resequenced in the current run; a tape file will require an extra pass whereas the direct access facility can be used to avoid this for the disk version.

THESE AND OTHER ARGUMENTS favor disk sequential processing; but on the other hand, tape sequential processing is often more economical.

1.11.6 INPUT MEDIA.

1.11.6.1 Console. The console typewriter will not be used as an input medium by application programs except where necessary for runtime conditions such as intervention on peripherals. Only conditions which cannot be determined prior to the job being read into the computer can be entered on the console typewriter. Approval must be obtained from Technical Evaluation & Standards Directorate (TESD) prior to the console being used as an input device.

1.11.6.2 Tape. All tape data files created by USACSC standard systems must employ standard labels as described in the appropriate vendor's reference manual.

1.11.6.3 Card. Card records will be defined as 80 character records. ASD's will employ card-to-tape or card-to-disk utilities for all large volume card inputs.

1.11.6.4 Direct Access Storage Devices. Direct access devices provide retrieval facilities for sequential, indexed, and direct file organization structures.

1.11.7 OUTPUT MEDIA.

1.11.7.1 Printer. Output data destined for printing will not be assigned by a program to a systems printer. Such a technique causes the program to run at the speed of the printer (a relatively slow unit record device) and makes program execution dependent upon the availability of the printer. The printer output will be assigned to the appropriate symbolic device depending on the target operating system. When the target operating system does not have device independence nor an automatic spool intercept capability, the printer output must be assigned to tape or disk depending on the hardware configuration (refer to USACSCM 18-2 series Executive Software Manual for Standardized SPOOL Utility interface specifications). The file descriptions must be designed to be compatible across these operating systems. Each report or listing not printed on preprinted special forms will be programed to provide the information requested below.

1.11.7.1.1 Security Classification. If the printed report is classified, the level of classification (CONFIDENTIAL, SECRET, etc.) will be centered at the top and bottom of each page. No other information will be on the classification lines. When programs which produce classified printed output are in a test status using other than live data, the classification will be replaced with UNCLASSIFIED which occupies the same number of positions as the actual classification. The downgrading of the classification will be programmatically controlled during the test phase. The marking and downgrading instructions contained in AR 380-5 will be used for production work.

1.11.7.1.2 First Header Line. The first header line of each page will contain, as the first entry at the extreme left of the line, the word "PREPARED" followed by the current date in the form of two numeric character day, three to nine alphabetic character month, two numeric character year. The title of the report will be centered on this line. The Report Control System (RCS) and the Product Control Number (PCN) will print following the title or at the extreme right of the line. The literal "PCN" and/or "RCS" will precede its associative data elements by one space (i.e., PCN AAA-A01, or RCS AAA-A01). If an as-of-date is required, it should appear following the RCS and/or PCN. As-of-date is categorized as constant data which varies by reporting cycle.

1.11.7.1.3 Remaining Header Lines. Remaining header lines will describe all columns of data that are contained in the report. The descriptions will be short words and/or meaningful abbreviations. Common descriptions within a system will be used.

1.11.7.1.4 Detail Lines. The actual report lines contained in the body of each page will be formatted according to the report specifications. When total lines are required within a report, the total line(s) should not be split across pages.

1.11.7.1.5 Page Line. The last line of each page or, in the case of classified reports, the next to the last line of each page, will contain the word PAGE to the right of the page followed by the page number. The final page of the report will contain the word END preceding the PAGE.

1.11.7.1.6 Spacing. The standard vertical spacing for all printed output will be six or eight lines per inch and the standard carriage control tape will be used for all printing. The use of non-standard carriage tapes (for use with forms other than standard 1413 or 11 inch deep paper) is not authorized. Special form spacing will be under program control.

1.11.7.1.7 Skipping. See FIGURE 1-2.

| CHANNEL | 6 PRINT LINES PER INCH | 8 PRINT LINES PER INCH |
|---|---|---|
| 1 | 5 and 71 (The first line of print of a page.) | 7th line (The first line of print of a page.) |
| 9 | 61 and 127 (Bottom of page print line for page number or next to last line of print if security classification is required.) | 81 (Bottom of page print line for page number or next to last line of print if security classification is required.) |
| 12 | Restricted for Remote Job Entry (RJE) terminal use. | |

FIGURE 1-2

1.11.7.2 Punch. Output data destined for punching will not be assigned by a program to a systems punch. Such a technique causes the program to run at the speed of the punch (a very slow unit record device) and makes program execution dependent upon the availability of the punch. The guidelines for the device assignment of punch output is the same as for the printer.

1.11.7.3 Tapes and Direct Access Storage Devices. Reference applicable subparagraphs under paragraph 1.11.6, Input Media.

1.11.8 PROGRAM TO OPERATOR MESSAGES. Program messages will be displayed upon either SYSLST or the console as follows.

1.11.8.1 Format. See FIGURE 1-3.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| XXXXXXXX | Ø<br>S | XX | Ø<br>S | X | Ø<br>S | X----X----X | |
| Program-<br>ID | P<br>A<br>C<br>E | Message<br>Number | P<br>A<br>C<br>E | Type<br>Message | P<br>A<br>C<br>E | Text up to 5Ø<br>positions | |

FIGURE 1-3

1.11.8.2 Program-ID. Will be the program name in accordance with Chapter 2 of TB 18-103, Software Design and Development.

1.11.8.3 Message Number. Messages will be numbered in sequence within each program. Messages numbered 1 through 9 will be preceded by a zero.

1.11.8.4 Type of Message. A one position code indicating the type of message. See FIGURE 1-4.

| CODE | MEANING |
|---|---|
| I | Information data as in run statistics. Print on printer only. IBM OS prints on SYSOUT; IBM DOS prints on SYSLIST. |
| D | Operator intervention required. |
| R | Error recovery message. |
| H | Programed halt unrecoverable. |

FIGURE 1-4

1.11.8.5 Halts. Halts will not be used by the programer except for conditions which prohibit continuation of a run (i.e., out of sequence files). Halt messages when used, shall be short and concise. All halts must be fully documented with stated recovery procedures for the system coordinator.

• The console operating instructions will specify the corrective processing actions required for each programed computer run halt. Following are examples of halt conditions:

1.11.8.5 Halts. (Cont.)

SEQUENCE ERROR HALTS

DATA CONSTANT CONTROL CARD HALTS

END-OF-JOB HALTS

OPERATING SYSTEM HALTS

OTHER PROGRAMED HALTS

UNPROGRAMED HALTS

1.11.9 RECOVERY GUIDANCE. Every program must have documented procedures describing the recovery operations required when a program is abnormally terminated. Program Check Point/Recovery standards and procedures are described in Chapter 2 of TB 18-103, Software Design and Development.

1.11.10 ERROR CONDITION OPTIONS. Error conditions disclosed as a result of auxiliary checking, i.e., cycle sequence validation on header labels, balancing, etc., must be programed to pause and wait for the operator's response to a programed message which allows for termination (error) or acceptance (continue computer run).

1.11.11 CONSOLE SWITCHES. Console switches will not be used since their use restricts the transferability of programs across vendor lines. Control/Parameter cards should be used when program operation is to be externally varied.

1.11.12 UTILITY PROGRAMS AND SUBROUTINES.

- For utility program usage and availability, reference the appropriate USACSCM 18-2 Executive Software Manual and/or the appropriate Vendors Utility Manual.

- For subroutine availability and usage, reference the appropriate USACSCM 18-2 Executive Software Manual.

# CHAPTER 2

# USACSC STANDARD PORTABLE EXPANDED COBOL

# ANSI COBOL SUBSET

2.1 ANSI COBOL. The American National Standard Common Business Oriented Language (ANSI COBOL) is the DA standard computer language for use in business-type information and data systems. USACSC represents the Department of the Army on the CODASYL COBOL Committee (formerly the Conference on Data Systems Languages Committee). Suggestions, additions or deletions concerning the Common Business Oriented Language (COBOL) should be addressed to:

DA CODASYL Representative
ACSC-TES

2.2 INTRODUCTION. In the past, program development has, to a large extent, been characterized by what can be called AUTONOMOUS growth. That is each system, at best, developed their own language concepts and standards, usually independent of any other system. In a Uniform Automated Data Processing environment this type of development leads to redundant effort, inconsistent guidelines and tends to compound the problems of programer training and program maintenance. A further ingredient that is missing, which is essential to a Standard Army Multicommand Management Information System (STAMMIS) environment, is the concept of central control. These standards have been developed in order to achieve the following objectives:

- Program Maintainability.
- Program Portability and Transferability.
- Programer Training and Understanding.
- Central Control (STAMMIS).
- Management Flexibility.

2.2.1 PURPOSE OF USACSC STANDARD PORTABLE EXPANDED COBOL, ANSI COBOL SUBSET SPECIFICATIONS. Chapter 2 is constructed to be a stand-alone document within the total presentation of the General Programing Standards. This recognizes that some readers of this manual are mainly concerned with the COBOL concepts. Therefore, the USACSC COBOL specifications contained in this chapter are intended to provide USACSC personnel, who are involved in STAMMIS development, with the rules of the language described in a source program environment. It is also the intent to provide, as much as possible, independence from hardware considerations. Therefore, this section provides the user with a convenient source for determining the syntax and behavior rules for the elements of COBOL.

## 2.3 LANGUAGE CONVENTIONS.

2.3.1 GLOSSARY. The definitions of the COBOL terms in the Glossary are provided merely as reference material or introductory material. The definitions are therefore brief and do not give any detail of syntactical rules. Most of the terms are further discussed in other areas of this manual. Refer to Attachment 1 for Glossary definitions.

2.3.2 LANGUAGE STRUCTURE. The individual characters of the language are concatenated to form character-strings and separators. A separator may be concatenated with another separator or with a character-string. A character-string may only be concatenated with a separator. The concatenation of character-strings and separators forms the text of a source program.

2.3.2.1 Separators. The space and punctuation characters, when not used as literals, are separators. More than one space may be used as a separator. The space may be used with other separators as defined as follows.

2.3.2.2 Punctuation Characters. In order for the punctuation characters period, comma and semicolon to be used as separators, a space must immediately follow.

2.3.2.3 Quotation Marks. When using quotation marks, a separator other than a quotation mark is required immediately to the left of an opening quotation mark and immediately to the right of a closing quotation mark.

2.3.2.4 Character-String. A character-string is a sequence of contiguous characters used to form a literal, a word, a PICTURE character-string, or a comment character-string.

2.3.2.5 Delimiters. A character-string is delimited by the separators: space, period, right parenthesis, comma and semicolon. A space must follow the period, comma or semicolon. A space must not immediately follow a left parenthesis nor immediately precede a right parenthesis. When using a PICTURE character-string, the separators used as delimiters are the space, period or semicolon. A space must not appear within the PICTURE character-string.

2.3.3 WORDS.

2.3.3.1 Definition of a Word. A word is made up of a combination of not more than 30 characters selected from the character set for words. The first or the last character cannot be a hyphen. The space is not an allowable character within a word; the space is a word separator.

2.3.3.2 Types of Words. There are two basic types of words: Reserved words and user-defined words. User-defined words must be distinct from all reserved words. For USACSC systems user-defined words must begin with an alphabetic character, must be unique and must not be qualified.

2.3.3.2.1 Reserved Words. Refer to Attachment 2 for a list of Reserved Words.

- A KEY WORD is a word that is required when the format in which it appears is used in a source program. In this manual, key words are upper-case and underlined.

- An OPTIONAL WORD is a word that is included in a format only to improve the readability. The presence or absence of an optional word does not alter the semantics of the format. These words appear in this manual as upper-case words that are not underlined.

- There are three types of CONNECTIVES: Qualifier connectives, series connectives and logical connectives.

The qualifier CONNECTIVES are used to associate a data-name or paragraph-name with its qualifier. The qualifier connectives are OF and IN.

The series CONNECTIVE is used to link two or more operands. The series connective is the comma.

The logical CONNECTIVES are used in compound conditions. The logical connectives are AND, OR, AND NOT and OR NOT.

- All SPECIAL REGISTERS are compiler-generated storage areas that are primarily used to store information produced with the use of specific COBOL features.

- All FIGURATIVE CONSTANTS are constants to which fixed data-names have been assigned. These data-names must not be inclosed in quotation marks when used as figurative constants. The singular and plural forms of figurative constants are equivalent and may be used interchangeably. Whenever a literal appears in a format, a figurative constant may be used in its place. There is one exception: If the literal is restricted to numeric characters, only the figurative constant ZERO (ZEROS, ZEROES) is allowed.

- The fixed DATA-NAMES and their meanings are as listed in FIGURE 2-1.

2.3.3.2.1 Reserved Words. (Cont.)

| | |
|---|---|
| ZERO<br>ZEROS<br>ZEROES | Represents one or more of the character of Ø depending on context. |
| SPACE<br>SPACES | Represents one or more blanks or spaces. |
| HIGH-VALUE<br>HIGH-VALUES | Represents one or more of the characters which are of the highest value in a computer's collating sequence. |
| LOW-VALUE<br>LOW-VALUES | Represents one or more of the characters which are of the lowest value in a computer's collating sequence. |
| QUOTE<br>QUOTES | Represents one or more of the character ". The word QUOTE (QUOTES) cannot be used in place of a quotation mark in a source program to inclose a non-numeric literal. |
| ALL literal | Represents one or more of the string of characters comprising the literal. The literal must be either a non-numeric literal or a figurative constant other than the ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only. |

FIGURE 2-1

• When a FIGURATIVE CONSTANT is used to represent a string of one or more characters, the length of the string is determined by the compiler from context according to the following rules:

When the FIGURATIVE CONSTANT is associated with another data name, as when the figurative constant is moved to or compared with another data item, the string of characters specified by the figurative constant is repeated character by character on the right until the size of the resultant string is equal to the size of the associated data item.

When the FIGURATIVE CONSTANT is not associated with another data item, such as when the figurative constant appears in a DISPLAY, the length of the string is one character. The figurative constant ALL literal may not be used with DISPLAY.

2.3.3.2.2 User-Defined Words.

• A LITERAL is a string of characters whose value is determined by the ordered set of characters of which the literal is composed. There are two types of literals, numeric and non-numeric.

A NUMERIC LITERAL is a string of characters selected from the digits Ø through 9, the plus sign, the minus sign and the decimal point. Numeric literals may be up to 18 characters in length. The rules for the formation of numeric literals are as follows:

A NUMERIC LITERAL must not contain more than one sign character. If a sign is used, it must appear as the leftmost character. If the literal is unsigned, it is assumed positive.

A NUMERIC LITERAL must not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere within the literal except as the rightmost character. If the literal does not contain a decimal point, it is considered to be an integer. The value of a numeric literal is the algebraic quantity represented by the characters in the numeric literal. Every numeric literal is category numeric. If the literal conforms to the rules for the formation of numeric literals, but is also inclosed in quotation marks, it is then considered to be a non-numeric literal by the compiler.

A NON-NUMERIC LITERAL is a string of allowable characters belonging to the USACSC character set, excluding the character quotation marks, bound by quotation marks. A non-numeric literal may be made up from 1 to 12Ø characters inclosed in quotation marks. Although IBM allows the apostrophe (single quote) to be used in lieu of quotation marks (double quotes), only the ANSI standard quotation marks (") will be used. Any spaces that are inclosed in the quotation marks are considered a part of the non-numeric literal, and therefore part of the value. Every non-numeric literal is placed in the category alphanumeric.

• A DATA-NAME is a word that contains at least one alphabetic character and does not begin with a hyphen and names an entry in the DATA DIVISION.

• A CONDITION-NAME is a word which is assigned to a specific value, set of values, or range of values within the complete set of values that a data-item may assume. The CONDITION-NAME is called a conditional variable.

• The CONDITION-NAME is used in conditions as an abbreviation for the relation condition. The relation condition assumes that the associated conditional variable is equal to one of the set of values to which that CONDITION-NAME is assigned.

2.3.3.2.2 User-Defined Words. (Cont.)

- A PROCEDURE-NAME is a word used to name a paragraph or a section which is used in the PROCEDURE DIVISION. A procedure-name may be composed solely of numerics, but is equivalent to another procedure-name only if they are both composed of the same number of digits and have the same numerical value.

- A MNEMONIC-NAME is used to assign a user-defined word to an implementor-name and must contain at least one alphabetic character. These associations are established in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION. This association allows the user to substitute the mnemonic-name for the implementor-defined name in any format where a substitution is valid.

2.3.4 CONCEPTS OF DATA REFERENCE.

2.3.4.1 Logical Record and File Concept. The approach of distinguishing between the physical aspects of the file and the conceptual characteristics of the data contained within the file is taken in defining file information.

The physical aspects of a file describe the data as it appears on the input or output media, such as mode in which the data file is recorded on the external medium, grouping of logical records within the physical limitations of the file medium, and a means by which a file can be identified.

The conceptual characteristics of a file are the definition of each logical entity within the file itself. The LOGICAL RECORD is a group of related information, uniquely identifiable and treated as a unit. The input and output statements pertain to one logical record.

A physical unit of information whose size and recording mode is convenient to a particular computer for the storage of data on an input or output device defines a PHYSICAL RECORD. The size of a PHYSICAL RECORD is dependent on the hardware and has no relationship with the size of the file of information contained on a device.

A physical unit may consist of one or more logical records. Also, in the case of mass storage files, a LOGICAL RECORD may require more than one physical unit to contain it. In this manual references to records refer to logical records, unless otherwise specified as physical records.

2.3.4.2 Concept of COBOL Levels. In structuring a LOGICAL RECORD, a level concept is inherent. This concept is necessary in order to specify subdivisions of a record. These subdivisions may be used for the purpose of data reference. Once a subdivision has been specified, it may be further subdivided in order to permit more detailed data reference.

2.3.4.2 Concept of COBOL Levels. (Cont.)

The elementary items are the most basic subdivisions. They cannot be further subdivided. A record consists of a sequence of elementary items or the record itself may be an ELEMENTARY ITEM.

A set of elementary items can be combined into a group. Each group is made up of a sequence of one or more elementary items. Groups may be combined together into other groups. Therefore, an ELEMENTARY ITEM may belong to more than one group.

2.3.4.3 Level-Numbers. A LEVEL-NUMBER is used to show the organization of an elementary item and a group item. All level-numbers for records start at Ø1, and less inclusive data items are assigned higher level-numbers not greater than 49.

A group includes all group and elementary items following it until a level-number less than, or equal to, the level-number of that group is encountered. All items which are immediately subordinate to a given group item must have level-numbers greater than the level-number of that given group item.

There are special LEVEL-NUMBERS 66, 77, and 88 which are exceptions to the above rules. There is no true concept of level for this entry.

The LEVEL-NUMBER 66 is used for entries which describe items by means of a RENAMES clause. This level-number is used for the purpose of regrouping data items. However, level-number 66 is not a part of USACSC COBOL.

The LEVEL-NUMBER 77 is used for entries that specify noncontiguous data items. These data items are not subdivisions of other items, nor are they themselves subdivided.

The LEVEL-NUMBER 88 is used for entries that specify condition-names. These entries are to be associated with particular values of a conditional variable.

2.3.4.4 Concept of Classes of Data. All data items are grouped into three classes: Alphabetic, numeric and alphanumeric.

For alphabetic and numeric, the class and the category of the data item are synonymous.

The class of alphanumeric includes the categories numeric-edited, alphanumeric-edited and alphanumeric (without editing).

Group level items are treated at object time as alphanumeric regardless of the class of the elementary items subordinate to that group item.

FIGURE 2-2 below shows the relationship of the class and categories of data items.

| Level of Item | Class | Category |
|---|---|---|
| Elementary | Alphabetic | Alphabetic |
| | Numeric | Numeric |
| | Alphanumeric | Numeric-Edited |
| | | Alphanumeric-Edited |
| | | Alphanumeric |
| Non-elementary (Group) | Alphanumeric | Alphabetic |
| | | Numeric |
| | | Numeric-Edited |
| | | Alphanumeric-Edited |
| | | Alphanumeric |

FIGURE 2-2

2.3.4.5 Algebraic Signs. There are two purposes for algebraic signs. They are used to show whether the value of an item involved in an operation is positive or negative and also used to identify an item for an edited report as being positive or negative.

The signs used with the item for an edited report are not operational signs, but are used only to display the sign of an item.

The editing signs are inserted into a data item through the use of the sign control symbols of the PICTURE clause.

The SIGN clause is used to state explicitly the location of the operational sign in a DISPLAY item. If the clause is not used, the operational sign is represented as defined by the implementor.

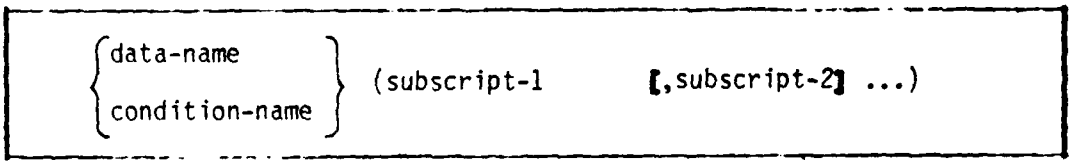
2.3.4.6 Qualification of Name. Every name in a COBOL source program must be unique. It must be made unique by making sure that no other name has the identical spelling and hyphenation. Qualification as a means of making a name unique is allowed only in conjunction with COPY libraries.

2.3.4.7 Subscripting. When there is a need to make reference to an individual element within a list or table of like elements that have not been assigned individual data-names, subscripting can be used.

An INTEGER or a DATA ITEM whose description is that of an integer may be used to represent a subscript. No subscript can itself be subscripted. The lowest possible value for a subscript is one (1). This is the first element in a table. The values 2, 3, etc. are used for the next sequential elements of that table. The highest permissible subscript value for any particular table is the maximum number of occurrences of the item that is specified in the OCCURS clause.

Parentheses are used to inclose the subscript or set of subscripts which follow the table element data-name which it identifies. When a table element data-name is appended by a subscript, it is called a subscripted data-name or an identifier. When more than one subscript is needed, the subscripts are written in the order of successively less inclusive dimensions of the data organization. A comma must be used to separate subscripts in a series.

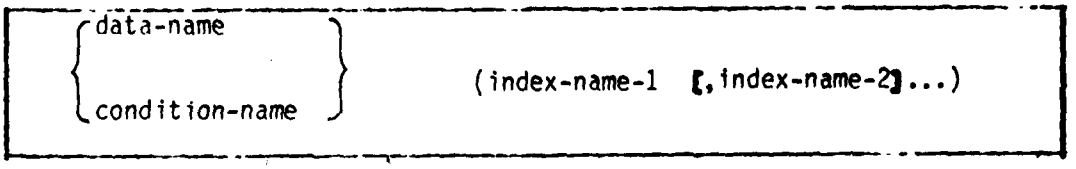
The general format for subscripting is:




2.3.4.8 Indexing. References to individual elements within a table of like elements can also be made by specifying indexing. This is done by using an INDEXED BY clause in the definition of a table. The name will be used to index the table and is referred to as an index data-name. This index data-name must be initialized by either a SET, SEARCH ALL or PERFORM statement before it can be used to reference the table.

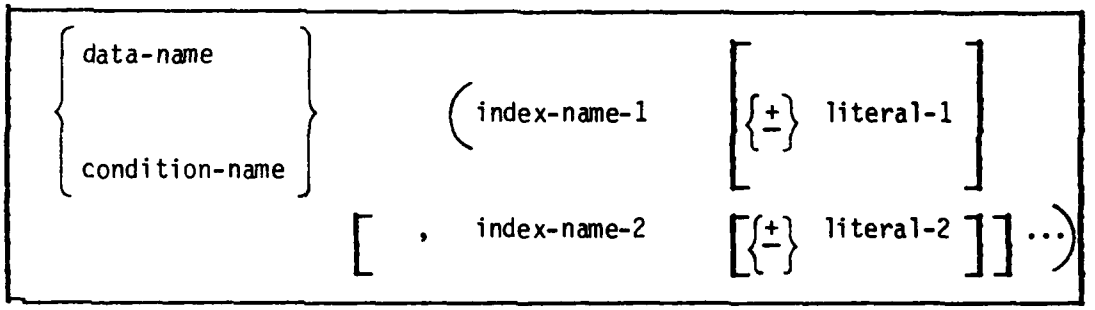
There are two kinds of indexing: direct indexing and relative indexing. Direct indexing is in the form of subscripting. Relative indexing is specified when an index data name is followed by a space, followed by one of the operators + or -, followed by another space, followed by an unsigned numeric literal, all inclosed within parentheses after the index data-name. When more than one index data-name is required, they are written in the order of successively less inclusive dimensions of the data organization. A comma may be used to separate indices in a series.

The general format for direct indexing is:



2.3.4.8 Indexing. (Cont.)

The general format for relative indexing is:




*For a further discussion of subscripting and indexing, see paragraph 2.5.4,* Table Handling Feature.

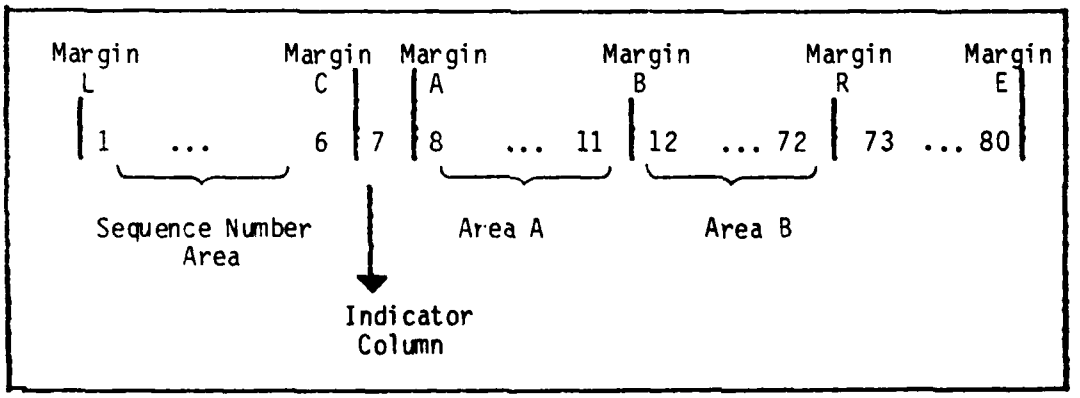
2.3.5 USACSC COBOL REFERENCE FORMAT.

2.3.5.1 General Description of Reference Format. The reference format is used to provide a standard method for describing COBOL source programs. The reference format is described in terms of character positions in a line on an input-output medium. The format consists of a standard 80-column punched card or 80-column line of which each of the 80 columns represents a character position. The COBOL compiler accepts source programs written in reference format and produces an output listing of the source program input in reference format.

2.3.5.1.1 Rules. The rules for spacing discussed in the reference format take precedence over all other rules for spacing.

2.3.5.1.2 Divisions. The divisions of a source program must be in the following order: the IDENTIFICATION DIVISION, the ENVIRONMENT DIVISION, the DATA DIVISION, then the PROCEDURE DIVISION. Each division must be written according to the rules for reference format.

2.3.5.2 Reference Format Representation. The reference format for a line is *represented as follows:*



2.3.5.2 Reference Format Representation. (Cont.)

Margin 'L' is immediately left of the leftmost character position of a line.

Margin 'C' is between the 6th and 7th character positions of a line.

Margin 'A' is between the 7th and 8th character positions of a line.

Margin 'B' is between the 11th and 12th character positions of a line.

Margin 'R' is between the 72nd and 73rd character positions of a line.

Margin 'E' is immediately right of the rightmost character position of a line.

The sequence number area occupies the first six (1-6) character positions of a line and is between Margin 'L' and Margin 'C'.

The indicator column is the 7th character position of a line. If there is no hyphen, it is assumed that the last character in the preceding line is followed by a space.

Area 'A' occupies character positions 8, 9, 10, and 11 beginning at Margin 'A' and ending at Margin 'B'.

Area 'B' occupies character positions 12 through 72 beginning at Margin 'B' and ending at Margin 'R'.

A program identification is placed in your source listing by the source library system in character positions 73 through 80 beginning at Margin 'R' and ending at Margin 'E', when extracting a source program from a Source Library System (SLS).

2.3.5.2.1 Sequence Numbers. A sequence number, consisting of six digits in the sequence number area, is used to numerically label each card image in a source program to be compiled by the COBOL compiler. The use of coded sequence numbers is optional since the USACSC SLS will automatically assign sequence numbers when extracting a source program. However, it is recommended that sequence numbers be assigned when coding. Sequence numbers should be incremented by 10.

2.3.5.2.2 Division Header. The first line in each division must be the division header. The division header starts at the left boundary of Area 'A' (character position 8). The division header consists of the division-name, followed by a space, the word DIVISION, and then a period. No other text may appear on the same line as the division header. However, if the program is to be called by another program, a space and a USING clause may follow the words PROCEDURE DIVISION.

2.3.5.2.3 Section Header. The section header must start at the left boundary of Area 'A'(character position 8). The section header consists of a section-name followed by a space, the word SECTION, and a period. If program segmentation is used, a space and a priority number followed by a period (.), may follow the word SECTION. No other text may appear on the same line as the section header, with the exception of the COPY sentences.

Sections appear in the ENVIRONMENT and PROCEDURE DIVISIONS and as DATA DIVISION entries in the DATA DIVISION.

2.3.5.2.4 Paragraph-Name and Paragraph

- A PARAGRAPH-NAME starts at the left boundary of Area 'A' (character position 8) of any line following the first line of a division or a section. A paragraph-name is followed by a period and a space.

- A PARAGRAPH consists of a paragraph-name followed by one or more sentences. The first sentence or entry in a paragraph begins on the next line following the paragraph-name at the left boundary of Area 'B' (character position 12); except in the IDENTIFICATION DIVISION and ENVIRONMENT DIVISION where the entry must be on the same line as the paragraph-name or in using the COPY statement which may also be on the same line as the paragraph-name. Each successive sentence in a paragraph must begin at the left boundary of Area 'B' (character position 12). When a sentence or entries of a paragraph require more than one line, they may be continued as described in paragraph 2.3.5.4.1, Continuation of Lines.

2.3.5.3 Data Division Entries. Each DATA DIVISION entry begins with a level indicator or a level-number, followed by a space, followed by a user-defined data-name or the reserved word FILLER, followed by a sequence of independent clauses describing the data item. The last clause is always terminated by a period followed by a space. There are two types of entries in the DATA DIVISION: those entries that begin with a level indicator and those entries that begin with a level-number.

2.3.5.3.1 Level Indicators. A LEVEL INDICATOR is either an FD (file description) or an SD (sort description). The level indicator begins at the left boundary of Area 'A', followed by the associated descriptive clauses. Those DATA DIVISION entries that begin with level-numbers are referred to as data description entries.

2.3.5.3.2 Data Description Entries. In a DATA DESCRIPTION ENTRY that begins with a level Ø1, the level-number begins at the left boundary of Area 'A' (character position 8), followed by two spaces, followed by the associated record-name or item-name, and then followed by its appropriate descriptive clauses. Level-numbers may be initially taken from a set of values Ø1 through 49 in increments of 2 (in order to allow for future insertions (i.e., Ø1, Ø3, Ø5, Ø7, etc.)) and the values 77 and 88.

2.3.5.3.3 Level-Numbers Ø1 through 49. These level-numbers are used to designate grouped data items that form records. Second level and successively increasing level-numbers begin in column 12 and must be indented four spaces over from the preceding level-number; except no level-numbers will go beyond column 24. The associated record-name follows the level-number by two spaces.

When a level-number of lower value than the preceding level-number is reached, a new group of data description entries is encountered. The levels for this new group of data description entries must then follow the above rules for successively increasing level-numbers.

2.3.5.3.4 Level-Number 77. This level-number is used for entries that describe independent data items. Level 77 entries must precede all other entries in the WORKING-STORAGE SECTION and LINKAGE SECTION and must begin at the left boundary of Area 'A' (character position 8), followed by the associated record-name or item-name in character position 12, and then followed by its appropriate descriptive clauses.

2.3.5.3.5 Level-Number 88. This level-number is used for entries that assign names to specific values that data items may assume. The level-number 88 must be coded in character positions 8 and 9, followed by its associated record-name or item-name in character position 12, and then followed by its appropriate descriptive clauses.

2.3.5.3.6 Clause. No more than one clause may be coded on any given card, with the exception of the first clause which may be coded on the first card with data-name or FILLER.

2.3.5.3.7 First Clause. The first clause, coded in a data description entry following the data-name or FILLER, may be coded no further to the left than character position 48 at the same (first) card. That clause may start to the right of column 48 if required in order to leave at least one space after data-name or FILLER. However, no clause may be placed on the first card if it cannot be completed on that card. If it cannot be completed on that first card, it must be placed on the second card and must follow the rules for the second and succeeding clauses as explained in the next paragraph.

2.3.5.3.8 Clauses. All clauses, coded on the second and following cards, must be coded beginning in character position 38, with continuation of literals beginning in character position 2Ø. (See paragraph 2.3.5.4.2, Continuation of Non-numeric Literals and paragraph 2.3.5.4.3, Continuation of Numeric Literals.)

2.3.5.4 Continuation.

2.3.5.4.1 Continuation of Lines. Any sentence or entry that requires more than one line is continued onto the next line and is started in character position 16. The subsequent lines are called the continuation line(s). The line being

2.3.5.4.1 Continuation of Lines. (Cont.)

continued is called the continued line. (This rule does not apply for DATA DIVISION entries.)

2.3.5.4.2 Continuation of Non-numeric Literals. A hyphen (-), placed in the indicator column (character position 7) of the continuation line, is used to indicate the continuation of a non-numeric literal. Prior to the continuation line, the last character of the non-numeric literal occurs in column 72. A quotation mark must be placed in character position 24 of the continuation line, followed by the continuation of the non-numeric literal. The non-numeric literal must be inclosed within quotation marks.

2.3.5.4.3 Continuation of Numeric Literals. A hyphen (-), placed in the indicator column (character position 7) of the continuation line, is also used to indicate the continuation of numeric literals. The numeric literal must begin on the continuation line in character position 20. The first non-blank character, starting at character position 20 of the continuation line, is to follow the last non-blank character of the continued line.

2.3.5.5 Blank Lines. A line is considered blank, if it contains nothing but spaces from the left boundary of Margin 'C' (character position 7) through the right boundary of Margin 'R' (character position 72). A blank line can appear anywhere within a source program, except immediately preceding a continuation line.

2.3.5.6 Comment Lines. An asterisk (*), placed in the indicator column (character position 7) of any line, is used to specify that that line is a comment. Any combination of characters may be used on a comment line. The asterisk and those characters will be printed on a source listing, but the comment line will not be used for any other purpose. In the Identification Division, comment lines are used to record a chronological history of program revisions. Each entry will include the revision number, revision date, and a brief narrative, in functional terms, of the program changes. Other comment entries are optional and may include program specifications, input specifications, and output specifications, name of the revisor and organization if different than the entries in AUTHOR and INSTALLATION.

On IBM equipment, Column 7 is also used as an indicator position in accordance with the multi-line coding concept. (Example: A=OS, B=DOS, etc.) See paragraph 2.8, SINGLE SOURCE LIBRARY SYSTEM.

2.3.6 USACSC STANDARD CODING CONVENTIONS. USACSC COBOL Coding Form 26 will be used when coding COBOL programs.

2.3.6.1 Coding. USACSC has developed several coding conventions which will be used to better enhance the COBOL language in the areas of readability. The following rules have been applied:

2.3.6.1.1 Symbols/Words. The symbols >, <, and = must not be used as relational operators. The words GREATER THAN, LESS THAN, and EQUAL, respectively, should instead be coded.

2.3.6.1.2 Verbs. Only one verb will be coded on each line.

2.3.6.1.3 Working-Storage Section. In organizing the WORKING-STORAGE area, the Ø level items should be placed in descending order according to frequency of use (i.e., the item with the highest frequency of use will be placed first, the item with the next highest frequency of use be placed second, etc.).

2.3.6.1.4 Paragraph/Section Names. All paragraph or section names will be on a separate line.

2.3.6.1.5 Coding Paragraph/Section Names. Paragraph and section names must be alphanumeric starting with four numeric characters followed by a hyphen, and they should also be descriptive of the statements contained under them.

2.3.6.1.6 Coding Data Description Entries. In the data description entries, when the following clauses are being used, they should be coded in this specific order as shown in FIGURE 2-3.

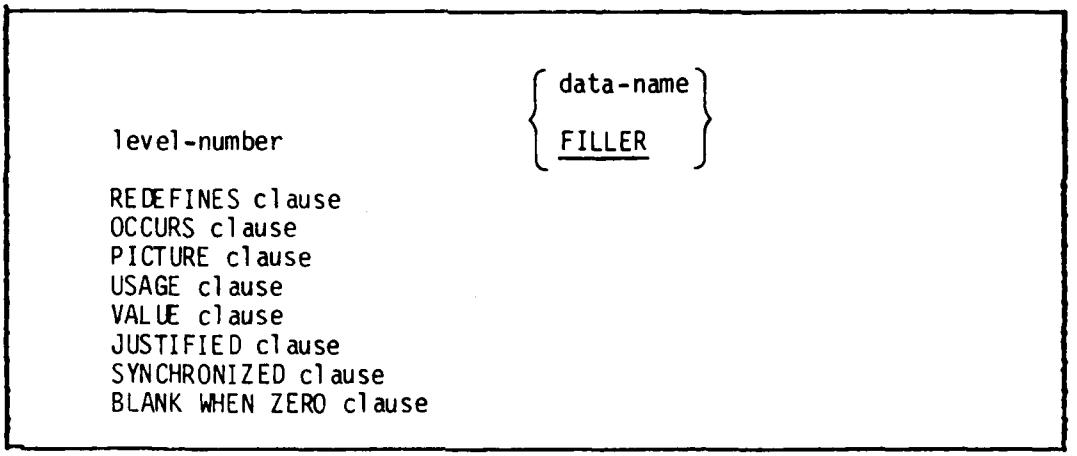



FIGURE 2-3

2.3.6.1.7 Clause. Each clause that follows an FD must begin in character position 12 of the next line. Each successive clause will begin on a new line.

2.3.6.1.8 Readability. In order to enhance readability, numeric literals should not be split. If the literal is too large, the whole VALUE clause should be placed on the next line. A special effort should be made to place a literal entirely on one line.

## 2.4 USACSC COBOL SPECIFICATIONS.

2.4.1 FORMAT RULES AND NOTES. The USACSC COBOL specifications will be presented in the following format:

2.4.1.1 Language Element. Identifies by name the particular COBOL language element specifications being dealt with. Only ANSI language element specifications will be used.

2.4.1.2 Function. This is a brief, general narrative description of the language element and its use.

2.4.1.3 Format. A general format defines the specific arrangement of the elements of a clause or statement. Formats are shown adjacent to information defining the clause or statement. When more than one specific arrangement is permitted, the general format is separated into numbered formats. Clauses must be written in the sequence given in the general formats. The format is graphically represented.

2.4.1.4 Syntax Rules. Syntax rules are those rules that define or clarify the order in which words or elements are arranged to form larger elements such as phrases, clauses or statements. Syntax rules also impose restrictions on individual words or elements. These rules are used to define or clarify how the statement must be written (i.e., the order of the elements of the statement and restrictions on what each element may represent).

2.4.1.5 General Rules. A general rule is a rule that defines or clarifies the meaning or relationship of meanings of an element or set of elements. It is used to define or clarify the semantics of the statement and the effect on either execution or compilation.

2.4.1.6 USACSC Guidelines. This will contain any additional guidance on usage by USACSC personnel.

2.4.2 FORMAT PUNCTUATION.

2.4.2.1 General Description. Punctuation characters comma and semicolon are shown in some formats. A semicolon, however, must not appear immediately preceding the first clause of an entry or paragraph. The use of these punctuation characters for each of the four divisions is noted as follows.

2.4.2.1.1 Identification Division. The comma and semicolon may be used within the comment-entries. The paragraph itself, however, must terminate with a period followed by a space.

2.4.2.1.2 Environment Division. Where either a comma or semicolon is shown in the formats, it is optional and may be included or omitted by the user. The entry itself must terminate with a period followed by a space.

2.4.2.1.3 Data Division. Where either a comma or semicolon is shown in the formats, it is optional and may be included or omitted by the user. The entry itself must terminate with a period followed by a space.

2.4.2.1.4 Procedure Division. When a comma or semicolon is shown in the formats, it is optional and may be included or omitted by the user. If desired, a semicolon may be used between statements within a paragraph or section.

2.4.2.2 Elements. Elements which make up clauses or statements consist of upper-case words, lower-case words, level-numbers, brackets, braces, connectives and special characters.

2.4.3 SYMBOLS AND NOTATIONS USED IN THIS MANUAL.

2.4.3.1 General. The various language elements that comprise a COBOL program must be written in formats that adhere to fixed and precise rules of presentation. It is necessary to understand the various symbols, rules and notations in describing the individual formats. Each format statement will indicate the following information:

The order of presentation.

Those words that are required for the proper functioning of the statement.

Those words that are optional and included at the user's discretion.

That information that must be supplied by the user.

Those elements in the statement that involve a choice by the user.

Those functions of a particular statement that are optional.

2.4.3.2 Format Presentation.

2.4.3.2.1 Words. All words inherent or built into COBOL are specified as upper-case.

2.4.3.2.2 Upper-Case Words (Underlined). All upper-case words which are underlined are required as key words. Those upper-case words not underlined are optional and may be used at the user's discretion.

2.4.3.2.3 Upper-Case Words (Underlined/Not Underlined). All upper-case words, whether underlined or not, are part of the COBOL language, and must be spelled exactly as indicated.

2.4.3.2.4 Lower-Case Words. All lower-case words are generic terms and must be supplied by the user.

2.4.3.2.5 Element Braces. Elements of a statement involving a required choice are surrounded by braces.

Example: { }

2.4.3.2.6 Function Brackets. Optional functions which may be included or omitted at the user's discretion are surrounded by brackets.

Example: [ ]

*Example:*

```
MULTIPLY { identifier-1 } BY identifier-2 [ROUNDED]
         { literal-1    }
      [ ; ON SIZE ERROR imperative statement]
```

2.4.3.3 Default Option. In some instances the choice can be made by default.

Example:

```
BLOCK CONTAINS [ integer-1 ] TO integer-2 { RECORDS    }
                                          { CHARACTERS }
```

The programer must choose either RECORDS or CHARACTERS. If RECORDS is chosen, the word RECORDS must be written because it is a key word (underlined). However, if CHARACTERS is the choice, CHARACTERS is not a key word and the programer need not write it. If the word CHARACTER is omitted, the COBOL compiler assumes that the word CHARACTER was chosen and generates machine code based on this assumption.

2.4.3.4 Ellipsis. In some statements, certain portions may be used as many times as needed by the programer. This repetition is indicated by the ellipsis (...). Brackets or braces are used as delimiters to indicate the portion of the statement which is repeatable.

2.4.3.4.1 Rules. The following rule applies to ellipsis:

Given an ellipsis (...) in a statement scan the statement from right to left beginning at the bracket or brace immediately to the left of the ... until the logically matching bracket or brace is found. The ... applies to the words within the logically matched brackets or braces.

2.4.3.4.2 Example 1:

```
ADD  { identifier-1 } [, identifier-2 ] ...  TO identifier-m [ROUNDED]
     { literal-1    } [, literal-2    ]
   [identifier-n [ROUNDED]] ...
   [; ON SIZE ERROR imperative-statement]
```

Scanning from right to left, starting at the bracket immediately to the left of the last ellipsis, it can be seen that the logically matching bracket is *the one preceding identifier-n. Thus the entire second line* of the statement can be written as many times as the programer chooses. Those brackets surrounding ROUNDED in both the first and second lines of the statement perform their normal function, i.e., they indicate which portion of the statement is optional.

2.4.3.4.3 Example 2:

```
MOVE  { identifier-1 }  TO identifier-2  [, identifier-3] ...
      { literal      }
```

Starting at the bracket immediately to the left of the ellipsis, the logically matching bracket is the bracket immediately preceding identifier-3. The programer may write as many different identifiers following the word TO

2.4.3.4.3 Example 2. (Cont.)

as he chooses. The braces in the statement perform their normal function; the programer must choose either identifier-1 or literal.

2.4.3.4.4 Usage. The preceding examples illustrate the usage of various elements of a COBOL statement. Certain language elements used in the examples (data-name, literal, identifier, imperative statement) are discussed in later sections.

2.4.4 COBOL PROGRAM STRUCTURE.

2.4.4.1 Divisions. Every COBOL source program is divided into four divisions. Each division must be placed in its proper sequence, and each must begin with a division header.

The four divisions, listed in sequence, and their functions are:

2.4.4.1.1 IDENTIFICATION DIVISION. This division names the program.

2.4.4.1.2 ENVIRONMENT DIVISION. This division indicates the machine equipment and equipment features to be used in the program.

2.4.4.1.3 DATA DIVISION. This division defines the nature and charactersitics of data to be processed.

2.4.4.1.4 PROCEDURE DIVISION. This division consists of statements directing the processing of data in a specified manner at execution time.

2.4.4.2 Formats. In all formats within this publication, the required clauses and optional clauses (when written) must appear in the sequence given in the format, unless the associated rules explicitly state otherwise.

2.4.4.3 Structure of the COBOL Program. Refer to FIGURE 2-4.

```
IDENTIFICATION DIVISION.          (Reference:  paragraph 2.4.5)
PROGRAM-ID.  program-name.
AUTHOR.  comment-entry.
INSTALLATION.  comment-entry.
DATE-WRITTEN.  comment-entry.
DATE-COMPILED.  comment-entry.
SECURITY.  comment-entry.

ENVIRONMENT DIVISION.             (Reference:  paragraph 2.4.6)
CONFIGURATION SECTION.
SOURCE-COMPUTER.  source-computer-entry.
OBJECT-COMPUTER.  object-computer-entry.
[SPECIAL-NAMES.  special-names-entry.]
[INPUT-OUTPUT SECTION.
FILE-CONTROL.  file-control-entry ...
[I-O-CONTROL.  I-O-control-entry]...]
```

FIGURE 2-4

2.4.4.3 Structure of the COBOL Program. (Cont.)

```
DATA DIVISION.                          (Reference:  paragraph 2.4.7)

[FILE SECTION.
 [FD - file-description-entry
        record-description-entry]...
          data-description-entry]
 [SD - sort-description-entry
        record-description-entry]...
          data-description-entry]]

[WORKING-STORAGE SECTION.
 [77-level-description-entry ] ...
 [record-description-entry   ]]

[LINKAGE SECTION.
 [77-level-description-entry]...
 [record-description-entry  ]]

[COMMUNICATION SECTION.
 [communication-description-entry [record-description-entry]...] ...]
PROCEDURE DIVISION.                     (Reference:  paragraph 2.4.8)

DECLARATIVES.

{section-name SECTION. USE Sentence
 [paragraph-name. [sentence]...]...}...

END DECLARATIVES.

{section-name SECTION [priority.]
 [paragraph-name. [sentence]...]...}...
```

FIGURE 2-4 (Cont.)

## 2.4.5 IDENTIFICATION DIVISION.

### 2.4.5.1 ELEMENTS.

FUNCTION. The IDENTIFICATION DIVISION identifies the source program and the resultant output listing. In addition, other documentation information may be supplied by the user in the pertinent paragraphs of this division.

FORMAT.

```
IDENTIFICATION DIVISION.

PROGRAM-ID. program-name.

AUTHOR. comment-entry.

INSTALLATION. comment-entry.

DATE-WRITTEN. comment-entry.

DATE-COMPILED. comment-entry.

SECURITY. comment-entry.
```

SYNTAX RULES.

- This division is always required.

- This division must start in the leftmost portion of Margin 'A' with the reserved words IDENTIFICATION DIVISION followed by a period and a space.

- All paragraph names are on a separate line and begin in the leftmost portion of Margin 'A'.

- The comment-entry may be any combination of characters from the computer's character set. The continuation of the comment-entry by the use of the hyphen in the indicator column is not permitted; however, the comment-entry may be contained on one or more lines.

GENERAL RULES. None.

VENDORS' GUIDELINES. IBM allows ID abbreviation.

USACSC GUIDELINES. The USACSC translator will permit the abbreviation ID for Identification Division.

2.4.5.2 PROGRAM-ID PARAGRAPH.

FUNCTION. The PROGRAM-ID paragraph gives the name by which a program is identified.

FORMAT.

```
PROGRAM-ID. program-name.
```

SYNTAX RULES. None.

GENERAL RULES.

- The PROGRAM-ID paragraph must contain the name of the program and must be present in every program.

- The program-name identifies the source program and all listings pertaining to a particular program.

VENDORS'GUIDELINES. The program-name must conform to the rules for formation of a system-name.

USACSC GUIDELINES. Program-names must be formatted as outlined in TB 18-103, Software Design and Development.

2.4.5.3 AUTHOR PARAGRAPH.

FUNCTION. The AUTHOR paragraph identifies the author or responsible programer.

FORMAT.

```
AUTHOR. comment-entry.
```

SYNTAX RULES.

- Begin in Margin 'A' and end with a period.

- The comment-entry may be any combination of characters from the computer's character set. The continuation of the comment-entry by the use of the hyphen in the indicator column is not permitted; however, the comment-entry may be contained on one or more lines.

GENERAL RULES. None.

USACSC GUIDELINES. Enter the programer name and/or office symbol within installation responsible for program; e.g., ACSC-TES-S.

2.4.5.4 INSTALLATION PARAGRAPH.

FUNCTION. The INSTALLATION paragraph identifies the responsible organization.

FORMAT.

```
INSTALLATION. comment-entry.
```

SYNTAX RULES.

- Begin at Margin 'A' and end with period.

- The comment-entry may be any combination of characters from the computer's character set. The continuation of the comment-entry by the use of the hyphen in the indicator column is not permitted; however, the comment-entry may be contained on one or more lines.

GENERAL RULES. None.

USACSC GUIDELINES. Code the organization symbol of the organization responsible for the maintenance of the program, i.e., ACSC-TES-S.

2.4.5.5 DATE-WRITTEN PARAGRAPH.

FUNCTION. The DATE-WRITTEN paragraph identifies when the program was written.

FORMAT.

```
DATE-WRITTEN. comment-entry.
```

SYNTAX RULES.

- Begin in Margin 'A', and end with a period.

- The comment-entry may be any combination of characters from the computer's character set. The continuation of the comment-entry by the use of the hyphen in the indicator column is not permitted; however, the comment-entry may be contained on one or more lines.

GENERAL RULES. None.

2.4.5.5 DATE-WRITTEN PARAGRAPH. (Cont.)

USACSC GUIDELINES. Code the date on which the IDENTIFICATION DIVISION of the first version of the program is written.

- Enter date format XX YYYYYYYYY ZZ

  XX = two numeric characters, current day.

  YYYYYYYYY = three to nine alpha characters for month.

  ZZ = two numeric characters, current year.

2.4.5.6 DATE-COMPILED PARAGRAPH.

FUNCTION. The DATE-COMPILED paragraph provides the compilation date in the IDENTIFICATION DIVISION source program listing.

FORMAT.

```
DATE-COMPILED. comment-entry.
```

SYNTAX RULES.

- Begin in Margin 'A', and end with a period.

- The comment-entry may be any combination of characters from the computer's character set. The continuation of the comment-entry by the use of the hyphen in the indicator column is not permitted; however, the comment-entry may be contained on one or more lines.

GENERAL RULES. The paragraph-name DATE-COMPILED causes the current date to be inserted during program compilation. If a DATE-COMPILED paragraph is present, it is replaced during compilation with a paragraph of the form:

DATE-COMPILED. current date.

USACSC GUIDELINES. Insert any comment entry; however, it will be replaced during compilation with the current date.

2.4.5.7 SECURITY PARAGRAPH.

FUNCTION. The SECURITY paragraph identifies the security classification of the source listing.

FORMAT.

```
SECURITY. comment-entry.
```

2.4.5.7 SECURITY PARAGRAPH. (Cont.)

SYNTAX RULES.

- Begin in Margin 'A', and end with a period.

- The classification comment-entry is a non-edited field that must end with a period. It should contain one of the Army-approved security classifications, i.e., unclassified, confidental, etc.

GENERAL RULES. None.

USACSC GUIDELINES. *Enter the appropriate security classification.*

2.4.5.8 REMARKS PARAGRAPH.

FUNCTION. The REMARKS paragraph is no longer supported in ANSI 74 COBOL. In order to facilitate conversion, an asterisk (*) is required in column 7. See paragraph 2.3.5.6, Comment lines.

2.4.6 ENVIRONMENT DIVISION.

2.4.6.1 ELEMENTS.

FUNCTION. Provides a standard way of expressing the computer dependent information needed to process a COBOL program.

FORMAT.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  source-computer-entry.
OBJECT-COMPUTER.  object-computer-entry.
[SPECIAL-NAMES.  special-names-entry.]
[INPUT-OUTPUT SECTION.
FILE-CONTROL.{file-control-entry}...  (Reference:  Paragraph 2.4.6.7)
[I-O-CONTROL.  I-O-control-entry]...]  (Reference:  Paragraph 2.4.6.14)
```

SYNTAX RULES. The Environment Division must begin in Area 'A' with the heading ENVIRONMENT DIVISION followed by a period.

2.4.6.1 ELEMENTS. (Cont.)

GENERAL RULES. The Environment Division is divided into two sections: the Configuration Section and the Input-Output Section. The sections and paragraphs, when written, must appear in the sequence shown in the above format.

USACSC GUIDELINES. This division is required for all USACSC programs. The USACSC translator will permit the abbreviation ED for ENVIRONMENT DIVISION.

2.4.6.2 CONFIGURATION SECTION.

FUNCTION. This section deals with the overall specifications of the computer.

FORMAT.

```
CONFIGURATION SECTION.
SOURCE-COMPUTER.  source-computer-entry.
OBJECT-COMPUTER.  object-computer-entry.
```

SYNTAX RULES. Section-names must begin at Margin 'A'.

GENERAL RULES. None.

USACSC GUIDELINES. The CONFIGURATION SECTION entry is required for all USACSC programs.

2.4.6.3 SOURCE-COMPUTER PARAGRAPH.

FUNCTION. The SOURCE-COMPUTER paragraph serves as documentation and describes the computer upon which the source program is compiled.

FORMAT.

```
SOURCE-COMPUTER.  source-computer-entry [WITH DEBUGGING MODE].
```

SYNTAX RULES. This paragraph begins in the A-margin.

GENERAL RULES.

- Computer-names are assigned by individual vendors.

2.4.6.3 SOURCE-COMPUTER PARAGRAPH. (Cont.)

- The WITH DEBUGGING MODE clause serves as a compile-time switch for the debugging statements written in the source program.

- When WITH DEBUGGING MODE is specified, all debugging sections and debugging lines are compiled.

- When WITH DEBUGGING MODE is omitted, all debugging sections and debugging lines are treated as documentation.

- The object-time switch dynamically activates the debugging code generated when WITH DEBUGGING MODE is specified.

- When debugging mode is specified, through the object-time switch, all the debugging sections and debugging lines compiled into the object program are activated.

- Recompilation of the source program is not required to activate or deactivate the object-time switch.

- When WITH DEBUGGING MODE is not specified in the SOURCE-COMPUTER paragraph, the object-time switch has no effect on execution of the object program.

- When debugging mode is suppressed, through the object-time switch, any USE FOR DEBUGGING declaration procedures are inhibited. However, all debugging lines remain in effect.

VENDORS' GUIDELINES. IBM requires the computer-name to be specified as either IBM-370-(model number) or IBM-360-(model number). For the 370, the model number is the form NNN, as 145, 155, etc. - for example, IBM-370-155. For the 360, the model number is in the form NN or ANN, where the NN is the numeric model designation, as 30, 40, etc., and the A is the memory-size designator, as G, H, I, etc. - examples, IBM-360-G40 or IBM-360-40. As shown, if the memory-size is not known, it need not be included in the model number.

USACSC GUIDELINES. This paragraph is required in all USACSC COBOL source programs.

2.4.6.4 OBJECT-COMPUTER PARAGRAPH.

FUNCTION. The OBJECT-COMPUTER paragraph describes the computer on which the program is to be executed.

2.4.6.4 OBJECT-COMPUTER PARAGRAPH. (Cont.)

FORMAT.

```
OBJECT-COMPUTER. object-computer-entry [ , MEMORY SIZE integer { WORDS      } ]
                                                               { CHARACTERS }
                                                               { MODULES    }
  [, PROGRAM COLLATING SEQUENCE IS alphabet-name]
  [, SEGMENT-LIMIT IS segment-number].
```

SYNTAX RULES. This paragraph begins at Margin 'A'.

GENERAL RULES.

- Computer-names are assigned by individual vendors.

- SEGMENT-LIMIT is used with the Segmentation facility. This option designates the highest segment-number for fixed permanent segments.

- The PROGRAM COLLATING SEQUENCE clause specifies that the collating sequence associated with alphabet-name is used in non-numeric comparisons.

USACSC GUIDELINES

- The object computer paragraph header is required in all USACSC source programs. The USACSC translator will provide the entry which will specify the lowest model machine on which program will be run.

2.4.6.5 SPECIAL-NAMES PARAGRAPH.

FUNCTION. The SPECIAL-NAMES paragraph provides a means of relating user-specified mnemonic-names to specific hardware devices or functions defined by each vendor.

2.4.6.5 SPECIAL-NAMES PARAGRAPH. (Cont.)

FORMAT.

```
SPECIAL-NAMES. [, implementor-name

{ IS mnemonic-name [ , ON STATUS IS condition-name-1 [ , OFF STATUS IS condition-name-2 ] ]
{ IS mnemonic-name [ , OFF STATUS IS condition-name-2 [ , ON STATUS IS condition-name-1 ] ]
{ ON STATUS IS condition-name-1 [,OFF STATUS IS condition-name-2]
{ OFF STATUS IS condition-name-2 [,ON STATUS IS condition-name-1] } } } ...

[ , alphabet-name IS { STANDARD-1 } ]
                     { NATIVE     }
                     { implementor-name }

[, CURRENCY SIGN IS literal-9]

[, DECIMAL-POINT IS COMMA].
```

SYNTAX RULES. A period must appear after the SPECIAL-NAMES paragraph title and after the last entry in the paragraph. Entries may be separated by spaces, commas, semicolons, but not periods.

GENERAL RULES.

- Implementor-name refers to specific features or devices defined by each vendor.
- Use of ON STATUS and OFF STATUS is vendor dependent.
- The alphabet-name clause provides a means for relating a name to a specified character code set and/or collating sequence. When alphabet-name is referenced in the PROGRAM COLLATING SEQUENCE clause or the COLLATING SEQUENCE phrase of a SORT or MERGE statement, the alphabet-name clause specifies a collating sequence.
- If the STANDARD-1 phrase is specified, the character code set or collating sequence identified is that defined in American National Standard Code for Information Interchange, X3.4-1968.

2.4.6.5 SPECIAL-NAMES PARAGRAPH. (Cont.)

• The CURRENCY SIGN IS clause specifies the literal that is used in the PICTURE clause to represent the currency symbol. The literal is limited to a single character and must not be one of the following:

digits Ø thru 9
alphabetic characters A, B, C, D, L, P, R, S, V, X, Z or the space
special characters '*', '+', '-', ',', '.', ';', '(', ')', '"', '/',
'='.

If the CURRENCY SIGN clause is not present only the $ can be used as the currency symbol ($) in the PICTURE clause.

• The clause DECIMAL-POINT IS COMMA means that the function of comma and period are exchanged in the character-string of the PICTURE clause and in numeric literals.

USACSC GUIDELINES.

• IBM.

• The following IBM carriage control characters which may be referred to by a mnemonic-name in the WRITE BEFORE/AFTER ADVANCING are:

CØ1 IS TOP-OF-PAGE
CØ9 IS BOTTOM-OF-PAGE
C12 is restricted to Remote Job Entry (RJE) terminal use.

(See FIGURE 1-2.)

• The USACSC translator will provide the vendor defined implementor-name for the target hardware.

• The following mnemonic names are USACSC standard carriage control names.

| FUNCTION-NAME | ACTION TAKEN |
|---|---|
| CSP | Sup[illegible] spacing |
| CØ1 through CØ9 | Skip [illegible] channel 1 through 9 respectively |
| C1Ø through C11 | Skip to channel 1Ø and 11 respectively |
| C12 | Reserved for RJE use |

2.4.6.6 INPUT-OUTPUT SECTION.

FUNCTION. Identifies each file and its external storage media, assigns input/output devices and supplies information needed for transmitting between external media and the object program.

2.4.6.6 INPUT-OUTPUT SECTION. (Cont.)

FORMAT.

```
[ INPUT-OUTPUT SECTION.                          ]
[ FILE-CONTROL.{file-control-entry}              ]
[ [I-O-CONTROL.  input-output-entry]...          ]
```

SYNTAX RULES. This paragraph begins at Margin 'A'.

GENERAL RULES. The INPUT-OUTPUT section is required if files are used, otherwise it is optional.

USACSC GUIDELINES. None.

2.4.6.7 FILE-CONTROL PARAGRAPH.

FUNCTION. The FILE-CONTROL paragraph names the file, identifies directly or indirectly the file medium and provides direct or indirect hardware device assignment.

FORMAT.

```
FILE-CONTROL.
     SELECT  [OPTIONAL] file-name
     ASSIGN TO implementor-name-1  [implementor-name-2] ...
     [; RESERVE integer-1 [AREA|AREAS]]
    ;ORGANIZATION IS    {SEQUENTIAL|INDEXED|RELATIVE}
    [;ACCESS MODE IS    {SEQUENTIAL  [, RELATIVE KEY IS data-name-1]
                        {RANDOM|DYNAMIC}, RELATIVE KEY IS data-name-1}]
     ; RECORD KEY IS data-name-1
    [; ALTERNATE RECORD KEY IS data-name-2 [WITH DUPLICATES]]...
    [; FILE STATUS IS data-name-3].
```

2.4.6.7 FILE-CONTROL PARAGRAPH. (Cont.)

SYNTAX RULES.

• The FILE-CONTROL paragraph is required when the INPUT-OUTPUT SECTION header is specified.

• The FILE-CONTROL paragraph header begins at Margin 'A', the clauses in Margin 'B'.

GENERAL RULES. None.

USACSC GUIDELINES. None.

2.4.6.8 SELECT CLAUSE.

FUNCTION. The SELECT clause is used to name each file in a program.

FORMAT.

```
SELECT [OPTIONAL] file-name

  ASSIGN TO implementor-name-1 [, implementor-name-2] ...

 [; RESERVE integer-1 [AREA ]]
                      [AREAS]

 ;ORGANIZATION IS    {SEQUENTIAL}
                     {INDEXED   }
                     {RELATIVE  }

[;ACCESS MODE IS     {SEQUENTIAL  [, RELATIVE KEY IS data-name-1]}
                     {RANDOM      , RELATIVE KEY IS data-name-1 }
                     {DYNAMIC                                   }]

 ; RECORD KEY IS data-name-1

[; ALTERNATE RECORD KEY IS data-name-2 [WITH DUPLICATES]]...

[; FILE STATUS IS data-name-3].
```

SYNTAX RULES.

• Each file named in the SELECT clause must have a file description entry (FD) or a sort description entry (SD) in the Data Division.

• Each file described in the Data Division must be named once and only once as a file name following the key word SELECT.

2.4.6.8 SELECT CLAUSE. (Cont.)

- When ACCESS and ORGANIZATION are not specified, SEQUENTIAL is implied.

GENERAL RULES. None.

VENDORS' GUIDELINES.

- IBM OS.

The file-name specified in the SELECT clause relates to the DD-name in the JCL entry for that file under OS.

Example: //ddname DD DSN=data set ...

OPTIONAL entry is taken as comments since this is a JCL function in OS.

- IBM DOS. The file-name specified in the SELECT statement relates to the file-name in TLBL or DLBL card of the DOS Job Control Language.

Example: //TLBL file-name ...
//DLBL file-name ...

USACSC GUIDELINES. None.

2.4.6.9 ASSIGN CLAUSE.

FUNCTION. Used to assign a file to an external medium.

FORMAT.

```
ASSIGN TO implementor-name-1 [,implementor-name-2] ...
```

SYNTAX RULES. The structure of the ASSIGN clause is dependent on each vendor's specifications.

GENERAL RULES. The ASSIGN clause is used to assign a file to an external medium. Each file used in the program must be referenced immediately after the SELECT statement and further referenced as a FD or SD entry in the DATA DIVISION.

VENDORS' GUIDELINES.

- IBM-CS. The ASSIGN clause is used to assign a file to an external medium.

2.4.6.9 ASSIGN CLAUSE. (Cont.)

OS FORMAT

```
ASSIGN TO [integer-1] system-name-1
          [system-name-2] ...
```

Integer-1 indicates the number of input/output units of a given medium assigned to file-name. However, since the number of units is automatically determined by the operating system, the integer-1 option need not be specified. When specified, it is treated as comments (see IBM System/360 Operation Syste...: Job Control Language, Form GC28-6539).

System-name specifies a device class, a particular input/output device, the organization of data upon this device, and the external-name of the file. All files used in a program must be assigned to an input/output medium. Any system-name beyond the first for a file will be treated as comments.

System-name has the following structure:

class -device -organization-name

Class is a 2-character field that specifies the device class:

DA (mass storage)
UT (utility)
UR (unit-record)

Files assigned to UT or UR must have standard sequential organization and can be accessed only sequentially. Files assigned to DA may have standard sequential or direct organization. When organization is direct, access may be either sequential or random.

Device is used to specify a particular device within a device class. It can be a 4 to 6-character field. If device independence for a file is desired, the device class must be UT; no device number may be specified. At execution time, such a file may be assigned to any device class (including unit-record).

The allowable system devices for any given class are as follows:

Mass storage (DA) 2301, 2302, 2303, 2311, 2314, 2321.
Utility (UT) 2301, 2302, 2311, 2314, 2321, 2400.
Unit-record (UR) 1403, 1404, (for continuous forms only),
1442R, 1442P, 1443, 1445, 2501, 2520R, 2520P, 2540R, 2540P.
(R indicates reader; P indicates punch.)

2.4.6.9 ASSIGN CLAUSE. (Cont.)

NOTE: Sort input, output, or work files may be assigned to any utility device except a 2321.

Program Product Information Version 3-OS.

For Version 3 only, the following additional system devices are allowable:

Mass Storage (DA) 2305-1, 2305-2, 2319, 3330.
Utility (UT) 2305-1, 2305-2, 2319, 3330.
Unit Record (UR) 3211.

NOTE: For the Version 1 and Version 2 Compilers, these devices (2305-1, 2305-2, 2319, 3330, or 3211) can be used, if the device field in system-name is omitted. At execution time, any of these devices can be specified through the UNIT subparameter of the file's DD statement. Note, however, that except for files containing spanned records the device field is treated as comments. For files containing spanned records, the block length for the file is checked against the maximum block length allowed for the device specified, and the smaller of the two becomes the block size that is used.

Program Product Information Version 4-OS.

The device field in system-name is treated as comments by the Version 4 Compiler. At execution time, any valid device can be specified through the UNIT subparameter of the file's DD statement. The following considerations apply:

If an invalid device number is specified, no error diagnostic is produced.

For an ASCII file, if 2400 (or other compatible tape device) is not specified in the device field, no error diagnostic is produced.

For a direct file with spanned records, the Version 4 Compiler always calculates buffer size from the COBOL record description.

Organization is a 1-character field that indicates the file organization. The following characters must be used:

S for files with standard sequential organization.
D for files with direct organization.

FIGURE 2-5 can be used to determine the correct choice for the organizational field in system-name.

2.4.6.9 ASSIGN CLAUSE. (Cont.)

Name is a 1 to 8-character field specifying the external-name by which the file is known to the system. It is the name that appears in the name field of the DD card for the file.

• IBM-DOS. The ASSIGN clause is used to assign a file to an external medium.

DOS FORMAT

```
ASSIGN TO [integer] system-name-1 [system-name-2] ...
```

Integer indicates the number of input/output units for a given medium assigned to file-name. Since the number of units is determined at program execution time (see IBM System/360 Disk Operating System; System Control and System Service Programs, Form C24-5036), the standard definition given above is not the action taken by this compiler. The above does not apply to sort work files which must be specified.

When specified for files with standard labels or for unlabeled output tape files, the integer option is treated as comments. When integer is specified as greater than one for unlabeled input tape files, then at the end of every reel a message is issued to the operator asking whether or not end-of-file has been reached. It is the user's responsibility to provide the operator with correct information as to the number of reels in the file.

For multivolume input files with nonstandard labels, the integer option is required. For such files, the compiler is unable to distinguish between end-of-volume and end-of-file and, therefore, cannot determine the number of reels in the file.

Therefore, for input files with nonstandard labels, the integer option is used to determine the number of reels in the file. If integer is not specified, the system assumes that the file is contained on one reel.

All files used in a program must be assigned to an external medium. System-name specifies a device class, a particular device, the organization of data upon this device, and the external name of the file. Any system-names beyond the first are treated as comments.

System-name has the following structure:

SYSnnn-class-device-organization-name

2.4.6.9 ASSIGN CLAUSE. (Cont.)

Where:

nnn is a 3-digit number between ØØØ and 221. This field represents the symbolic unit to which the file is assigned.

class is a 2-digit field that represents the device class. The allowable combinations of characters are:

DA (mass storage)
UT (utility)
UR (unit record)

Files assigned to DA devices may have standard sequential or direct organization. When organization is direct, access may be either sequential or random.

Files assigned to UT or UR devices must have standard sequential organization.

device is a 4 or 5-digit field that represents a device number. Device number is used to specify a particular device within a device class.

The allowable devices for any given device class are as follows:

Mass storage (DA) 2311, 2314, 2321.
Utility (UT) 24ØØ, 2311, 2314, 2321.
Unit record (UR) 1442R, 1442P, 14Ø3, 14Ø4.
(continuous forms only), 1443, 25Ø1, 252ØR, 252ØP, 254ØR, 254ØP.
(R indicates reader, P indicates punch)

NOTE: Sort input, output, or work files may be assigned to any utility device except a 2321.

Organization is a 1-character field that specifies file organization. The letters that may be specified for each type of file are as follows:

S for standard sequential files.
A for direct files -- actual track addressing.
D for direct files -- relative track addressing.

FIGURE 2-6 can be used to determine the correct choice of the organization field in system-names.

name is a 1 to 7-character field specifying the external-name by which the file is known to the system. If specified, it is the name that

2.4.6.9 ASSIGN CLAUSE. (Cont.)

appears in the file-name field of the VOL, DLAB, TPLAB, DLBL, or TLBL job control statement. If name is not specified, the symbolic unit (SYSnnn) is used as the external name. The field must be specified if more than one file is assigned to the same symbolic unit.

Program Product Information -- Version 3 DOS.

For Version 3, the following additional system devices are allowable:

Mass Storage (DA) 2319, 3330.
Utility (UT) 2319, 3330, 3410, 3420.
Unit Record (UR) 3211, 3505, 3525P, 3525R, 3525W, 3525M.

For the Version 3 DA and UT devices (2319, 3330, 3410, 3420), as well as for the UR 150-character printer (3211), these numbers can be specified in the device field of system-name. For these devices, the valid entries for the other fields in system-name are unchanged. For the 3505 card reader, system-name has the following format:

SYSnnn-UR-3505- { S R / 0 } [ -name ]

The SYSnnn and name fields have the same valid entries as other devices.

For the organization field, the following considerations apply:

S R specifies standard sequential card reading. The optional R field specifies RCE (Read Column Eliminate) card reading. When R is specified, the user can indicate by program control that some card columns are to be ignored when reading data for a particular job. (See the section "RCE and OMR Format Descriptor" for a more complete discussion). When the R field is omitted, RCE card reading may not be specified.

0 specifies Optical Mark Reading (OMR). When 0 is specified, then if at object time the device reads a card with a marginal mark, a wear

2.4.6.9 ASSIGN CLAUSE. (Cont.)

mark, or a poor erasure, the substitution character (hexadecimal "3F") is placed in the defective column and in column 80 (an 80-character buffer is always provided). (Also, see the section "RCE and CMR Format Descriptor" for a further discussion.)

For the 3525 card punch with special features, a system-name has the following format:

```
                  { P  }                 { S R }
SYSnn-UR-3525     { R  }   -             { V R }      [-name]
                  {    }                 { X R }
                  { W  }                 {     }
                  { M  }                 { Y R }
                                         { T   }
                                         { Z   }
```

NOTE: The optional R code in the organization field is valid only when the device is specified as 3525R.

The name field has the same valid entries as for other devices.

The SYSnnn field, for 3525 files that do not utilize combined function processing, has the same valid entries as other devices.

The SYSnnn field has special considerations when combined function card processing is used. For each associated logical file within the combined function structure there must be a separate SELECT sentence; each such associated logical file must be specified with the same SYSnnn field.

For the device field, the following entries are valid:

3525R for a card read file.
3525P for a card punch.
3525W for a 2-line card print file.
3525M for a multiline card print file.

For the organization field, depending on the device field, the following entries are valid:

| | | |
|---|---|---|
| 3525R | S R | for sequential card read files. |
| (reader) | V R | for read/print associated files. |
| | X R | for read/punch/print associated files. |
| | Y R | for read/punch associated files. |

2.4.6.9 ASSIGN CLAUSE. (Cont.)

Note: the optional R field specifies RCE (Read Column Eliminate) card reading. (See "RCE and OMR Format Descriptor" for further discussion.)

| | |
|---|---|
| 3525P (punch) | S for sequential card punch files.<br>T for punch-and-interpret files (see Note).<br>X for read/punch/print associated files.<br>Y for read/punch associated files.<br>Z for punch/print associated files. |

NOTE: The T field denotes a normal punched output file for which the graphically printable punched characters are also printed on print lines 1 and 3 of the card. Line 1 contains the first 64 characters, left justified; line 3 contains the last 16 characters, right justified.

| | |
|---|---|
| 3525W (2-line print) | S for sequential 2-line print files.<br>V for read/print associated files.<br>X for read/punch/print associated files.<br>Z for punch/print associated files. |
| 3525M (multi-line print) | S for sequential multiline print files.<br>V for read/print associated files.<br>X for read/punch/print associated files.<br>Z for punch/print associated files. |

- RCE AND OMR FORMAT DESCRIPTOR.

When the user specifies O (for Optical Mark Read) or R (for Read Column Eliminate) in the organization field of system-name, then at object time he must provide a format descriptor as the first card(s) in his data deck. If the format descriptor is missing for such files, a message is issued to the operator, and the job is terminated.

The format descriptor must be the first card(s) in the data deck. Column 1 of the first card must be blank. The keyword FORMAT must be punched in columns 2 through 7. Column 8 must be blank. Columns 9 through 71 can contain the parameters that specify which columns of the data cards are to be read in OMR or RCE mode. Continuation cards are valid. A continuation code must be placed in column 72 of the preceding card. Parameters may then be continued, beginning in column 16 of the continuation card. Comments, if used, must follow the last operand on each card by at least one blank space, and continuation card restrictions must be observed.

2.4.6.9 ASSIGN CLAUSE. (Cont.)

The format of the format descriptor is as follows:

```
Col.
12....7.9.............
11     1 1
11     1 1
11     1 1
VV     V V
 FORMAT (N1, N2)    ,(N3, N4) ...
```

N1, N2, N3, and N4 may be any decimal integers from 1 through 8Ø. However, N2 must be greater than or equal to N1. N4 must be greater than or equal to N3. In addition, for OMR processing, N1 and N2 must be both even or both odd, N3 and N4 must be both even or both odd, and N3 - N2 must be greater than or equal to 2.

In OMR mode, the user establishes which columns are to be read in OMR mode. For example, if the user wishes to read columns 1, 3, 5, 7, 9 and 7Ø, 72, 74, 76, 78, 8Ø in OMR mode, the following format descriptor is valid:

```
FORMAT (1,9), (7Ø,8Ø)
```

In RCE mode, the user specifies those columns which are not to be read. For example, if the user chooses to eliminate columns 2Ø through 3Ø, and columns 52 through 73, the following format descriptor is valid:

```
FORMAT (2Ø,3Ø), (52,73)
```

2.4.6.9 ASSIGN CLAUSE. (Cont.)

FIGURE 2-5 below identifies the values of organization field for file organization for version 4-OS

| Device Type | ACCESS | File Organization | Track Addressing | Organization Field in System-name |
|---|---|---|---|---|
| tape, punch, reader, printer | SEQUENTIAL | standard sequential | -- | S |
| mass storage device | SEQUENTIAL | standard sequential | -- | S |
| mass storage device | SEQUENTIAL | direct | relative | D |
| mass storage device | RANDOM | direct | relative | D |
| mass storage device | RANDOM | direct (REWRITE) | relative | W |
| mass storage device | SEQUENTIAL | indexed | -- | I |

FIGURE 2-5

2.4.6.9 ASSIGN CLAUSE. (Cont.)

FIGURE 2-6 below identifies the values of organization field for Version 3 DOS.

| Device Type | ACCESS | File Organization | Organization Field |
|---|---|---|---|
| UR and UT (except 35Ø5, 3525) | SEQUENTIAL | standard sequential | S |
| UR 35Ø5, 3525R (without OMR or RCE) | SEQUENTIAL | standard sequential | S |
| UR 35Ø5 (with OMR) | SEQUENTIAL | standard sequential | O |
| UR 35Ø5, 3525R (with RCE) | SEQUENTIAL | standard sequential | SR |
| UR 3525R, 3525P, 3525W, 3525M | SEQUENTIAL | standard sequential | S |
| UR 3525P punch-interpret file | SEQUENTIAL | standard sequential | T |
| UR 3525R, 3525W 3525M read/print associated file | SEQUENTIAL | standard sequential | V |
| UR 3525R (with RCE) read/print associated file | SEQUENTIAL | standard sequential | VR |
| UR 3525R, 3525P, 3525W, 3525M read/punch/print associated file | SEQUENTIAL | standard sequential | X |
| UR 3525R (with RCE) read/punch/print associated file | SEQUENTIAL | standard sequential | XR |

FIGURE 2-6

2.4.6.9 ASSIGN CLAUSE. (Cont.)

| Device Type | ACCESS | File Organization | Organization Field |
|---|---|---|---|
| UR 3525R, 3525P read/punch associated file | SEQUENTIAL | standard sequential | Y |
| UR 3525R (with RCE) read/punch associated file | SEQUENTIAL | standard sequential | YR |
| UR 3525P, 3525W 3525M punch/print associated file | SEQUENTIAL | standard sequential | Z |
| DA (mass storage) devices | Entries valid for Version 2 are valid for Version 3 | | |

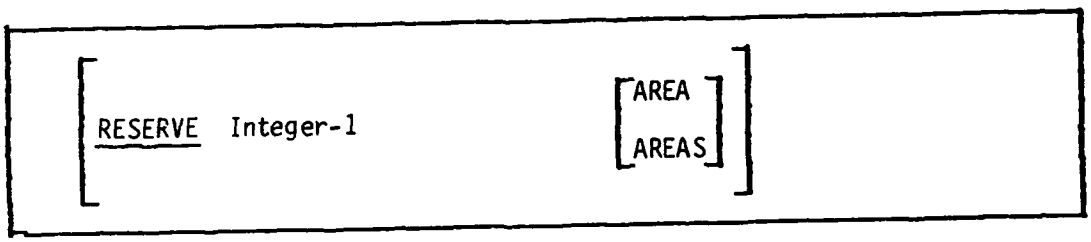
FIGURE 2-6 (Cont.)

USACSC GUIDELINES. For use on current USACSC IBM equipment, the procedure is to use a single source coding with an OS baseline. To create a particular DOS or OS object module an extract is made from the source library system comparing column 7 for an appropriate DOS or OS code. See Single Source Library System procedures at paragraph 2.8.

2.4.6.10 RESERVE CLAUSE.

FUNCTION. The RESERVE clause allows the user to modify the number of input/output areas (buffers) allocated by the computer.

FORMAT.




SYNTAX RULES. The value of integer-1 must not exceed 254.

GENERAL RULES.

- If the RESERVE clause is not specified, the number of input/output areas allocated is specified by the implementor.

VENDORS' GUIDELINES.

- IBM OS.

If RESERVE and SAME AREA clause are omitted, the number of buffers assigned are taken from the DD card. If RESERVE and SAME AREA clause are omitted and no buffers are specified in the DD card, two areas are reserved.

- IBM DOS.

A minimum of one buffer is required for a file. If this clause is omitted or if one is specified, one additional buffer is assumed.

If the RESERVE clause is not specified, no additional buffer areas are reserved aside from the minimum of one.

This clause may be specified only for a file whose organization is standard sequential.

USACSC GUIDELINES. None.

2.4.6.11 ACCESS CLAUSE.

FUNCTION. The ACCESS clause defines the manner in which records of a file are to be accessed.

2.4.6.11 ACCESS CLAUSE. (Cont.)

FORMAT.

```
ACCESS MODE IS { SEQUENTIAL }
               { RANDOM     }
               { DYNAMIC    }
```

SYNTAX RULES. Sequential access is assumed if clause omitted.

GENERAL RULES.

- Sequential access may be applied to files residing on tape, unit record, or mass storage.

- For random access, file must be assigned to mass storage and retrieval is based upon RELATIVE KEY associated with each record.

- For dynamic access, records in the file may be accessed sequentially and/or randomly.

VENDORS' GUIDELINES.

- IBM.

For ACCESS IS RANDOM, storage and retrieval are on the basis of a RELATIVE or NOMINAL KEY associated with each record. When the RANDOM option is specified, the file must be assigned to a mass storage device. ACCESS IS RANDOM may be specified when file organization is direct, relative, or indexed.

USACSC GUIDELINES. None.

2.4.6.12 RELATIVE KEY CLAUSE.

FUNCTION. The clause identifies a data-name that can be directly used by the system to locate a logical record on a mass storage device.

FORMAT.

```
RELATIVE KEY IS data-name-1
```

SYNTAX RULES.

- The RELATIVE KEY clause is affected by/affects execution of the READ, REWRITE, START and WRITE statements.

2.4.6.12 RELATIVE KEY CLAUSE. (Cont.)

GENERAL RULES.

- Must be specified for direct files when ACCESS is RANDOM.

- Value of data-name-1 must be established prior to READ or WRITE.

- Data-name-1 must be defined in the File, Working-Storage, or Linkage Section. However, if data-name-1 is specified in the File Section, it may not be contained in the file for which it is the key.

- If the RELATIVE KEY phrase is specified, execution of the READ statement is dependent upon the FORMAT used.

- FORMAT 1 updates the contents of the RELATIVE KEY data item such that it contains the relative record number of the record made available.

- Execution of a FORMAT 2 READ statement sets the current record pointer to, and makes available, the record whose relative record number is contained in the data item named in the RELATIVE KEY phrase for the file. If the file does not contain such a record, the INVALID KEY condition exists and execution of the READ statement is unsuccessful.

- For a REWRITE of a file accessed in either random or dynamic access mode, the MSCS logically replaces the record specified by the contents of the RELATIVE KEY data item associated with the file. If the file does not contain the record specified by the key, the INVALID KEY condition exists; the updating operation does not take place and the data in the record area is unaffected.

- For the START statement, a comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file-name and a data item referenced by the RELATIVE KEY clause associated with file-name.

  If the comparison is satisfied, the current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

  If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, the execution of the START statement is unsuccessful, and the position of the current record pointer is undefined.

- For the WRITE statement, when a file is opened in the output mode, records may be placed into the file by one of the following:

  If the access mode is sequential, the WRITE statement will cause a record to be released to the MSCS. The first record will have a relative record number of one (1) and subsequent records released will have relative record numbers of 2, 3, 4, ... . If the RELATIVE KEY data item has been specified in the file control entry for the associated file, the relative record number of the record just released will be placed into the RELATIVE KEY data item by the MSCS during execution of the WRITE statement.

2.4.6.12 RELATIVE KEY CLAUSE. (Cont.)

If the access mode is random or dynamic, prior to the execution of the WRITE statement, the value of the RELATIVE KEY data item must be initialized in the program with the relative record number to be associated with the record in the record area. That record is then released to the MSCS by execution of the WRITE statement.

When a file is opened in the I-O mode and the access mode is random or dynamic, records are to be inserted in the associated file. The value of the RELATIVE KEY data item must be initialized by the program with the relative record number to be associated with the record in the record area. Execution of a WRITE statement then causes the contents of the record area to be released to the MSCS.

VENDORS' GUIDELINES.

- IBM.

- Data-name-1 may be from 5 to 259 bytes in length.

- The first four bytes of data-name-1 are the track identifier and must be defined as a 5-integer binary data item whose maximum value does not exceed 65,535.

- The remainder of data-name-1 -- 1 through 255 bytes in length -- represents the record identifier. It is the user's responsibility to select from 1 through 255 bytes for the symbolic portion of the RELATIVE KEY field.

USACSC GUIDELINES. Reference is directed to USACSCM Executive Software Module P66ATP (DOS) and P59ATU (OS).

2.4.6.13 RECORD KEY CLAUSE.

FUNCTION. A RECORD KEY is used to access an indexed file. It specifies the item within the data record that contains the key for the record.

FORMAT.

```
RECORD KEY IS data-name-1
```

SYNTAX RULES.

- Data-name-1 may be any fixed-length item within the record. It must be less than 256 bytes in length.

- When two or more record descriptions are associated with a file, a similar field must appear in each description, and must be in the same relative position from the beginning of the record, although the same data-name-1 must not be used for both fields.

2.4.6.13 RECORD KEY CLAUSE. (Cont.)

- Data-name-1 must be defined to exclude the first byte of the record in the following cases:

  Files with unblocked records.

  Files from which records are to be deleted.

  Files whose keys might start with a delete-code character (HIGH VALUE).

- With these exceptions, the item specified by data-name-1 may appear anywhere within the record.

GENERAL RULES. The RECORD KEY clause must be specified for an indexed file.

USACSC GUIDELINES. None.

2.4.6.14 ALTERNATE RECORD KEY CLAUSE.

FUNCTION. An ALTERNATE RECORD KEY clause specifies a record key that is an alternate record key for the file. This alternate record key provides an alternate access path to records in an indexed file.

FORMAT.

```
ALTERNATE RECORD KEY IS data-name-2 [WITH DUPLICATES] ...
```

SYNTAX RULES.

- The value of an alternate record key may be non-unique if the DUPLICATES phrase is specified for it.

GENERAL RULES.

- The data description of data-name-2 and its relative location within a record must be the same as that used when the file was created.

- The number of alternate keys for the file must also be the same as that used when the file was created.

USACSC GUIDELINES. None.

2.4.6.15 FILE STATUS CLAUSE.

FUNCTION. The FILE STATUS clause is used to indicate to the COBOL program the status of an input-output operation.

2.4.6.15 FILE STATUS CLAUSE. (Cont.)

FORMAT.

```
[; FILE STATUS is data-name-3]
```

SYNTAX RULES.

- Data-name-3 may be qualified.

GENERAL RULES.

- When the FILE STATUS clause is specified, a value will be moved by the operating system into the data item specified by data-name-3 after the execution of every statement that references that file either explicitly or implicitly. This value indicates the status of execution of the statement.

2.4.6.16 I-O CONTROL PARAGRAPH.

FUNCTION. The I-O CONTROL paragraph defines some of the special control techniques to be used in the object program. This paragraph specifies the points at which rerun is to be established, the memory area which is to be shared by different files, and the location of files on a multiple file reel.

FORMAT.

```
I-O CONTROL.

[;RERUN [ ON { file-name-1 } ] EVERY { [END OF] { REEL } OF file-name-2 } ... ]
                { implementor-name }   { integer-1 RECORDS }
                                       { integer-2 CLOCK-UNITS }
                                       { condition-name }
[;SAME [SORT] [RECORD] AREA FOR file-name-3 { , file-name-4 } ... ] ...
[;MULTIPLE FILE TAPE CONTAINS file-name 5 [POSITION integer-3]
    [, file-name-6 [POSITION integer-4]] ... ] ...
```

2.4.6.16 I-O CONTROL PARAGRAPH. (Cont.)

SYNTAX RULES.

- The I-O CONTROL paragraph is optional.

- File-name-1 must be a sequentially organized file.

- The END OF REEL/UNIT clause may only be used if file-name-2 is a sequentially organized file. The definition of UNIT is determined by each implementor.

- When either the integer-1 RECORDS clause or the integer-2 CLOCK-UNITS clause is specified, implementor-name must be given in the RERUN clause.

- More than one RERUN clause may be specified for a given file-name-2, subject to the following restriction: When multiple integer-1 RECORDS clauses are specified or when multiple END OF REEL or END OF UNIT clauses are specified, no two of them may specify the same file-name-2.

- Only one RERUN clause containing the CLOCK-UNITS clause may be specified.

- The two forms of the SAME clause are considered separately in the following: More than one SAME clause may be included in a program, however a file-name must not appear in more than one SAME AREA clause.

- The files referenced in the SAME AREA clause need not all have the same organization or access.

GENERAL RULES.

- The RERUN clause specifies when and where the rerun information is recorded. Rerun information is recorded in the following ways:

  a. If file-name-1 is specified, the rerun information is written on each reel or unit of an output file and the implementor specifies where, on the reel or file, the rerun information is to be recorded.

  b. If implementor-name is specified, the rerun information is written as a separate file on a device specified by the implementor.

- There are many forms of the RERUN clause, based on the several conditions under which rerun points can be established. The implementor must provide at least one of the specified forms of the RERUN clause.

  a. When either the END OF REEL or END OF UNIT clause is used without the ON clause. In this case, the rerun information is written on file-name-2, which must be an output file.

2.4.6.16 I-O CONTROL PARAGRAPH. (Cont.)

b. When either the END OF REEL or END OF UNIT clause is used and file-name-1 is specified on the ON clause. In this case, the rerun information is written on file-name-1, which must be an output file. In addition, normal reel, or unit, closing functions for file-name-2 are performed. File-name-2 may either be an input or an output file.

c. When either the END OF REEL or END OF UNIT clause is used and implementor-name is specified in the ON clause. In this case, the rerun information is written on a separate rerun unit defined by the implementor. File-name-2 may be either an input or output file.

d. When the integer-1 RECORDS clause is used. In this case, the rerun information is written on the device specified by implementor-name, which must be specified in the ON clause, whenever integer-1 records of file-name-2 have been processed. File-name-2 may be either an input or output file with any organization or access.

e. When the integer-2 CLOCK-UNITS clause is used. In this case, the rerun information is written on the device specified by implementor-name, which must be specified in the ON clause, whenever an interval of time, calculated by an internal clock, has elapsed.

f. When the condition-name clause is used and implementor-name is specified in the ON clause. In this case, the rerun information is written on the device specified by implementor-name whenever a switch assumes a particular status as specified by condition-name. In this case, the associated switch must be defined in the SPECIAL-NAMES paragraph of the Configuration Section of the Environment Division. The implementor specifies when the switch status is interrogated.

g. When the condition-name clause is used and file-name-1 is specified in the ON clause. In this case, the rerun information is written on file-name-1, which must be an output file, whenever a switch assumed a particular status as specified by condition-name. In this case, as in paragraph f above, the associated switch must be defined in the SPECIAL-NAMES paragraph of the Configuration Section of the Environment Division. The implementor specifies when the switch status is interrogated.

- The SAME AREA clause specifies that two or more files that do not represent sort or merge files are to use the same memory area during processing. The area being shared includes all storage area assigned to the files open at the same time.

USACSC GUIDELINES. None.

2.4.6.17 RERUN CLAUSE.

FUNCTION. The RERUN clause specifies the intervals at which the checkpoint records are to be taken.

FORMAT.

```
[ ;RERUN [ ON { file-name-1        } ] EVERY { { [END OF] { REEL } }                  OF file-name-2 } ... ]
              { implementor name   }       { { UNIT }      }
                                           {   integer-1 RECORDS }
                                           { integer-2 CLOCK-UNITS }
                                           { condition-name }
```

SYNTAX RULES. File-name-2 represents the file for which the checkpoint records are to be written. It must be described with a file description entry in the Data Division.

GENERAL RULES. Checkpoint records are written sequentially and assigned to mass or tape storage devices.

USACSC GUIDELINES.

• The need for recovery must be considered in all system designs. The types of recovery required by a subsystem will depend on such factors as frequency of operation, history records that must be maintained, the speed at which the subsystem must recover and required data accuracy. Every system designer must consider recovery procedures and document how recovery operations will be accomplished. Any generalized approach to recovery is inefficient and difficult to use. Each system designer must, therefore develop his own recovery procedures. A checkpoint will be taken at least every sixty minutes of program execution time. The RERUN clause is not always adequate for use in restarting complete systems. Specialized restart procedures should also be allowed in lieu of checkpoints (e.g., job or step restart), when appropriate.

• Checkpoint Considerations. The system analyst must give consideration to the following points:

Available I/O device for writing the checkpoint, i.e., use unassigned unit if possible.

Select the input file that is close to sixty minutes per reel in running time and take a checkpoint each time this reel ends.

If the fastest input file requires more than sixty minutes to reach end of volume, it will be necessary for the programer to use the number of records option, i.e., take checkpoint every 9,999 records.

2.4.6.17 RERUN CLAUSE. (Cont.)

• Program Checkpoint Considerations. When a program is expected to run for longer than sixty minutes, provision should be made for taking checkpoint information periodically during the run. This information describes the status of the job and the system (main storage, input/output status, general and floating point registers) at the time the records are written. This provides a means of restarting at a checkpoint position rather than at the beginning of the entire job if processing is terminated for any reason before the normal end-of-job. To permit taking checkpoints and restarting, the programer should insure that:

Upon restarting, the program will be able to continue as though it has just reached that point in the program at which termination occurred.

File handling is organized to permit easy reconstruction of the status of the system as it exists at the time of each checkpoint.

The contents of files are not altered significantly between the time of the checkpoint and the time of the restart in sequential files. All records written on the file at checkpoint time should be unaltered at restart time. With non-sequential files, care must be taken to design the program so that a restart will not duplicate work that has been completed between checkpoint and restart time.

• Restart Procedures. Restart procedures must list, for the operator, the actions required to recover that unit of work or series of jobs. This may constitute restarting the program or may require that previous programs be rerun to recreate input files. Complete reference to programs and files necessary to accomplish the rerun must be given as well as any manual actions to be taken, when applicable, prior to rerunning the program.

• Device Error Recovery. Most I/O devices have a unique device error recovery routine. The appropriate routine is entered upon detection of an error. In all these routines an attempt is made to recover from the error. Additional error recovery may be attempted by programing or by operator action. The following choices are available:

An error can be ignored (record processed).

The job can be terminated.

The problem program can take action (an exit to a user routine is allowed).

The record in error can be by-passed.

2.4.6.18 SAME CLAUSE.

FUNCTION. The SAME clause specifies that two or more files are to use the same storage area during processing.

2.4.6.18 SAME CLAUSE. (Cont.)

FORMAT.

```
[;SAME [SORT / RECORD] AREA FOR file-name-3 {,file-name-4} ...] ...
```

SYNTAX RULES. For file-name-3, file-name-4, etc., enter the names of the appropriate files the same way they are coded in the SELECT statement.

GENERAL RULES.

- A specific file-name must not appear in more than one SAME AREA clause.
- A specific file-name must not appear in more than one SAME RECORD AREA clause.
- If one or more file-name of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in that SAME RECORD AREA clause. However, that SAME RECORD AREA clause may contain additional file-names that do not appear in that SAME AREA clause.
- File names specified in a given SAME clause may not be open simultaneously.

USACSC GUIDELINES. In IBM DOS COBOL, SAME RECORD AREA should not be used. It increases core requirements unnecessarily and the same effect can be achieved by using one WORKING-STORAGE entry for all files with the same Record Description. WORKING-STORAGE entries are easier to locate in a core dump than a compiler assigned work area.

2.4.6.19 MULTIPLE FILE TAPE CLAUSE.

FUNCTION. MULTIPLE FILE TAPE clause is used to indicate that more than one file occurs on a tape reel.

FORMAT.

```
[;MULTIPLE FILE TAPE CONTAINS file-name-5
 [POSITION integer-3] [;file-name-6 [POSITION integer-4]] ...] ...
```

2.4.6.19 MULTIPLE FILE TAPE CLAUSE. (Cont.)

SYNTAX RULES. Integer indicates the position of the file relative to the beginning.

GENERAL RULES.

- The MULTIPLE FILE clause is required when more than one file shares a single physical reel of tape.
- Regardless of the number of files on a reel, only those files that are used in a program need be specified.
- If all the file-names have been listed in consecutive order, the POSITION clause need not be given.
- If any file in the sequence is not listed, the POSITION clause must be listed.
- No more than one file on the same reel can be open at one time.
- Tape files with RECORDING MODE ASCII or EBCDIC must be named in the MULTIPLE FILE clause.

USACSC GUIDELINES. None.

2.4.7 DATA DIVISION.

2.4.7.1 ELEMENTS.

FUNCTION. The DATA DIVISION describes the information that is processed by the object program. This includes the data that is accepted as input, created, manipulated, or produced as output. Data to be processed falls into the following categories:

- Data contained in files input to the object program or output from it. This data enters or leaves the internal memory of the computer from a specified computer storage device.
- Data developed internally and placed in an intermediate (working) storage area.
- Constants defined by the program in an intermediate (working) storage area.
- Linkage data descriptions used for communication between the main program and subprograms.

2.4.7.1 ELEMENTS. (Cont.)

FORMAT.

```
DATA DIVISION.
[FILE SECTION.
    [FD - file-description-entry
          record-description-entry] ...
          data-description-entry]
    [SD - sort-description-entry
          record-description-entry] ...
          data-decription-entry ]
[WORKING-STORAGE SECTION.
    [77-level-description-entry     ] ...
    [record-description-entry       ]
[LINKAGE SECTION.
    [77-level-description-entry     ] ...
    [record-description-entry       ]
[COMMUNICATION SECTION.
    [communication-description-entry [record-description-entry]...] ...]
```

SYNTAX RULES.

- DATA DIVISION must begin in Area 'A'.
- Each section is optional.

GENERAL RULES. Within each section may be found different levels of entries. The record description is the highest level of organization in all sections except the File Section where it is subordinate to the file description. The data description is subordinate to the record description. The entries are differentiated by the use of a level indicator, level-number or special level-number.

- The level indicator (FD or SD) is used only in the File Section to specify the beginning of a file description or sort description entry. It must be followed by at least one record description entry and its associated data description entries.

2.4.7.1 ELEMENTS. (Cont.)

• The level-number is used to structure a record so that individual elements of data within the record may be referenced. The level-number represents either a record description or a data description.

The level-number Ø1 indicates a record description entry. This is the highest level of data organization and is not subordinate to any other items of data.

Data descriptions are subdivisions of record descriptions and have a level-number higher than Ø1. For record descriptions which are not subdivided, the record description and the data description are synonymous. Data descriptions may be further subdivided as long as each level of sub-division has a numerically higher level-number assigned. (See paragraph 2.3.5, USACSC COBOL Reference Format for formatting standards for data descriptions.)

The lowest subdivision of a record is an elementary item. The elementary item entry contains a level-number, one or more spaces, the data item-name, and a period. The size of the group item is computed from the sizes of its elementary items. Group items may be subdivisions of other group items. The group item contains all the subordinate group items and elementary items with level-numbers higher than the level-number of that group item.

• The special level-number does not structure a record. It represents a special type entry.

The data item description entry is used only in the WORKING-STORAGE SECTION and is prefixed by a special level-number of 77. This is an elementary data item which is not a subdivision of a group item and cannot be subdivided.

The condition-name entry is prefixed by a special level-number of 88. This entry indicates a specific value or range of values a data item might contain during program execution.

USACSC GUIDELINES. None.

2.4.7.2 FILE SECTION.

FUNCTION. The FILE SECTION contains the descriptions of all data stored externally (input and output files) and of all sort-files used in the program.

2.4.7.2 FILE SECTION. (Cont.)

FORMAT.

```
[ FILE SECTION.
[FD - file-description-entry
        record-description-entry ]...
[          data-description-entry]
[SD - sort-description-entry
        record-description-entry ]...
[          data-description-entry]  ]
```

SYNTAX RULES. The FILE SECTION begins with the header FILE SECTION followed by a period. This section contains file description and sort description entries followed by their associated record description entries.

GENERAL RULES.

- FILE/SORT DESCRIPTIONS.

In a COBOL program the File Description (FD) and/or Sort File Description (SD) represents the highest level of organization in the FILE SECTION. The FILE SECTION header is followed by one or more FD entries consisting of a level indicator (FD or SD), a data-name, and a series of independent clauses. These clauses specify the size of the logical and physical records, the recording mode and label information and the names of the data records which comprise the file. The entry is terminated by a period and followed by record descriptions associated with it.

A Sort-File Description (SD) gives information about the name and size of data records in the work file. The name Sort-File designates a set of records sorted by a SORT statement. There are no label procedures which the user can control and the rules for blocking and internal storage are peculiar to the SORT statement.

USACSC GUIDELINES.

- DATA-NAMES. Unique data-names must be assigned to each data item in a program to eliminate the necessity for qualification. Use as COBOL data-names the abbreviation specified on the appropriate DA Form 4738, contained in the Program Maintenance Manual (PM). A prefix may be attached to the beginning of a standard data-name to distinguish an item in a master record from the same item in a transaction. For data-names which are not in AR 18-12, use a meaningful description. Source statement library facilities will be used for record descriptions occurring in more than two programs provided they are supported by the compiler and the operating system.

2.4.7.2 FILE SECTION. (Cont.)

The use of standard Data Division entries without recoding them is available through the library copy feature. These entries will be stored in user-created libraries, and included in the source program at compile time. The levels used by a COPY clause and the level of the data-name being copied from the library must be the same.

To avoid data-name qualification, the data-names under each file description may be prefixed. The prefix should be meaningful to the developer according to program requirements. The resultant prefix will uniquely identify any data-name to the FD and appropriate Ø1-level record description.

- Improved processing can be achieved by placing the FD's in the order of most used to least used files, and also placing the most used working storage areas first.

2.4.7.3 WORKING-STORAGE SECTION.

FUNCTION. The WORKING-STORAGE SECTION describes data items and records that are developed and processed internally. It is composed of the section header, followed by data item description entries (77-level independent data items) and record description entries (Ø1-level) in that order.

FORMAT.

```
[ WORKING-STORAGE SECTION.                          ]
[                                                   ]
[   [77-level-description-entries]  ...             ]
[   [record-description-entry     ]                 ]
[                                                   ]
```

SYNTAX RULES. The section must begin with the section header WORKING-STORAGE SECTION followed by a period.

GENERAL RULES. The section contains data description entries for non-contiguous items (77-level) and record description entries (Ø1-level) in that order.

USACSC GUIDELINES.

- Each independent data item-name and each record name must be unique. Data-names subordinate to the record-name must also be unique.

- When data-names are not described by CSCM 18-5 or AR 18-12, data-names used will clearly describe the function of the field.

2.4.7.3 WORKING-STORAGE SECTION. (Cont.)

- All elementary numeric items used as a subscript or as the object of an OCCURS ... DEPENDING ON or a GO TO .... DEPENDING ON clause should be defined as COMPUTATIONAL. This will preclude the addition of extra code by the compiler to convert the number to binary each time it is referenced.

- The character 'S' indicating an operational sign is strongly recommended for inclusion in the PICTURE clause of every numeric elementary item with a USAGE of COMPUTATIONAL.

- Numeric items with an explicit or implicit USAGE of DISPLAY should have a PICTURE with a sign indicator of 'S' if arithmetic is to be performed on this field repeatedly.

- Working storage entries will be prefixed with "WS-" or "WSnn-". The only exception being those data items and record-names used in conjunction with COPY library which are excluded from the WS or WSnn prefix requirement.

For small-to-average-sized programs where there is little or no problem of multiple occurrences of the same data element in different record descriptions, prefix all data-names with the characters "WS-" in the WORKING-STORAGE SECTION. See FIGURE 2-7.

For programs that are very large and/or which have large areas of WORKING-STORAGE, use the following rules for the WORKING-STORAGE SECTION. See FIGURE 2-7.

Example of use in small-to-average-sized programs:

```
WORKING-STORAGE SECTION.
77  WS-PAGE-COUNTER                PIC S999
                                USAGE IS COMP   (NOTE:  COMP usage
                                                 varies by vendor)
                                VALUE IS ZERO.
Ø1  WS-CARD-AREA.
    Ø3  WS-RECORD-ID                PIC X.
    Ø3  FILLER                      PIC X(7Ø).
    Ø3  WS-NAME-FIELD               PIC X(9).
Ø1  WS-TRANSACTION-ID               PIC X.
```

FIGURE 2-7

2.4.7.3 WORKING-STORAGE SECTION. (Cont.)

Example of use in small-to-average-sized programs: (Cont.)

```
Ø1  WS-TRANSACTION-ID                    PIC X
                                       VALUE IS "A".
Ø1  WS-CORRECTION-ID                     PIC X
                                       VALUE IS "C".
```

Example of use in large programs:

```
WORKING-STORAGE SECTION.
Ø1  WSØ1-PAGE-COUNTER                    PIC S999
                                       USAGE IS COMP
                                       VALUE IS ZERO.
Ø1  WSØ1-END-OF-FILE-SWITCH              PIC X.
Ø1  WSØ1-NAME-HOLD-AREA                  PIC X(9).

Ø1  WSØ2-CARD-ARE-1.
    Ø3  WSØ2-RECORD-ID                   PIC X.
    Ø3  FILLER                           PIC X(79).
Ø1  WSØ3-CARD-AREA-2                   REDEFINES WSØ2-CARD-AREA-1.
    Ø3  WSØ3-RECORD-ID                   PIC X.
    Ø3  FILLER                           PIC X(7Ø).
    Ø3  WSØ3-NAME-FIELD                  PIC X(9).
Ø1  WS-TRANSACTION-ID                    PIC X
                                       VALUE IS "A".
Ø1  WS-CORRECTION-ID                     PIC X
                                       VALUE IS "C".
```

FIGURE 2-7 (Cont.)

1. Prefix all fixed-value constants in WORKING-STORAGE SECTION with the characters "WS-" and group them all together in one contiguous area in WORKING-STORAGE, preferably in the back of the section. These data elements would be used in place of literals in the body of the program's PROCEDURE DIVISION, etc.

2. Prefix all variable value data elements in WORKING-STORAGE SECTION with characters of the construction "WSnn-", where the nn is normally (but not always, as will be explained later) a two-digit numeric designator that uniquely identifies either a group of independent variables and small record description entries where there are no problems of repetition of the same data element in more than one place, or individual record description entries (Ø1-level entries). The areas, of either type discussed above, should be designated in ascending numeric sequence, thus making any portion of WORKING-STORAGE easy to find in the program listing.

2.4.7.3 WORKING-STORAGE SECTION. (Cont.)

- All elementary numeric items used as a subscript or as the object of an OCCURS ... DEPENDING ON or a GO TO .... DEPENDING ON clause should be defined as COMPUTATIONAL. This will preclude the addition of extra code by the compiler to convert the number to binary each time it is referenced.

- The character 'S' indicating an operational sign is strongly recommended for inclusion in the PICTURE clause of every numeric elementary item with a USAGE of COMPUTATIONAL.

- Numeric items with an explicit or implicit USAGE of DISPLAY should have a PICTURE with a sign indicator of 'S' if arithmetic is to be performed on this field repeatedly.

- Working storage entries will be prefixed with "WS-" or "WSnn-". The only exception being those data items and record-names used in conjunction with COPY library which are excluded from the WS or WSnn prefix requirement.

For small-to-average-sized programs where there is little or no problem of multiple occurrences of the same data element in different record descriptions, prefix all data-names with the characters "WS-" in the WORKING-STORAGE SECTION. See FIGURE 2-7.

For programs that are very large and/or which have large areas of WORKING-STORAGE, use the following rules for the WORKING-STORAGE SECTION. See FIGURE 2-7.

Example of use in small-to-average-sized programs:

```
WORKING-STORAGE SECTION.
77  WS-PAGE-COUNTER                PIC S099
                                USAGE IS COMP   (NOTE:  COMP usage
                                                 varies by vendor)
                                VALUE IS ZERO.
Ø1  WS-CARD-AREA.
    Ø3  WS-RECORD-ID                PIC X.
    Ø3  FILLER                      PIC X(7Ø).
    Ø3  WS-NAME-FIELD               PIC X(9).
Ø1  WS-TRANSACTION-ID               PIC X.
```

FIGURE 2-7

2.4.7.3 WORKING-STORAGE SECTION. (Cont.)

Use of this convention allows the same data element to appear in many different locations without having to do the one thing that will most destroy a program's usability and maintainability -- making minor changes to the meaningful portion of the data-name in order to make it unique, thus destroying the continuity of the program where that data element is concerned.

For the larger programs, with large amounts of storage required, the use of the dual method of "WSnn-" variables and "WS-" constants has numerous advantages. These include:

1. Easy access to any portion of a long listing of storage through use of the numeric sequence.

2. Elimination of the problem of data-name uniqueness where the same data element occurs in multiple locations.

3. Unique, easily recognizable labeling of any area of storage through the designation of blocks of numbers to a given series of record types. For example, all report output record areas in WORKING-STORAGE might be set up to carry prefixes in the 40's; all input transaction record areas prefixes in the 50's; etc.

4. An extremely high number of areas may be set up using this method. Simply within the normal numeric series, up to 99 different blocks of data elements may be established. For extremely large programs, though, or those with an extremely high number of record description entries, the following extension to the "WSnn-" prefix is suggested. Once the entire 99 numbers have been employed, begin using a prefix of the structure "WSxn-", where the x is an alphabetic character, beginning with the letter "A" and working up, with the n character, still a numeric digit, working through the sequence from zero (Ø) to nine (9) for each alphabetic character. Thus the first area would be prefixed "WSAØ-", the second "WSA1-", the third "WSA2-", etc. This method adds 26Ø more prefix combinations to the available set, certainly covering any foreseeable situations.

5. Using the dual method, the appearance of a "WS-" type prefix immediately labels that data-name to the programer as belonging to a constant whose value will not change. The programer doesn't have to constantly keep referring to the listing to determine the type of data element being worked with.

The "WSnn-" data-name prefixing applies to new programs under development. Existing programs will be converted to these standards mechanically through the use of USACSC COBOL formatting programs when and as convenient.

2.4.7.4 LINKAGE SECTION.

FUNCTION. The Linkage Section is used to describe data made available in a called program from a calling program. See VENDORS' GUIDELINES for additional features available under IBM Operating System.

FORMAT.

```
[ LINKAGE SECTION.                                    ]
[                                                     ]
[ [77-level-description-entry ]  ...                  ]
[ [record-description-entry   ]                       ]
[                                                     ]
```

SYNTAX RULES. The LINKAGE SECTION begins with the header LINKAGE SECTION followed by a period.

GENERAL RULES. Data item description entries and record description entries in the Linkage Section provide names and descriptions, but storage within the program is not reserved since the data area exists in a calling program. All data description clauses may be used to describe items in the Linkage Section, with one exception: the VALUE clause may not be specified for other than level-88 items.

VENDORS' GUIDELINES.

- The Linkage Section, under the IBM Operating System (OS), is also used to describe data from the PARM field of the EXEC statement. Data is made available to a main program at execution time.

- In the Linkage Section, the compiler assumes that each level-Ø1 item starts on a doubleword boundary.

- The combined total number of level-77 and level-Ø1 items in the Linkage Section may not exceed 255.

USACSC GUIDELINES. Linkage Section entries will be prefixed with "LS-" or "LSnn". The only exception being when COPY library is used in conjunction with the Linkage Section.

- For small-to-average-sized programs where there is little or no problem of multiple occurrences of the same data element in different record descriptions prefix all data names with the characters "LS-" in the LINKAGE SECTION.

- For programs that are very large and/or which have large LINKAGE area, use the following rules for the LINKAGE SECTION.

2.4.7.4 LINKAGE SECTION. (Cont.)

Prefix all fixed-value constants in the LINKAGE SECTION with the characters "LS-" and group them all together in one contiguous area, preferably in the back of the section. These data elements would be used in place of literals in the body of the program's PROCEDURE DIVISION, etc.

Prefix all variable value data elements in the LINKAGE SECTION with characters of the construction "LSnn-", where the nn is normally (but not always, as will be explained later) a two-digit numeric designator that uniquely identifies either a group of independent variables and small record description entries where there are no problems of repetition of the same data element in more than one place, or individual record description entries (Ø1-level entries). The areas, of either type discussed above, should be designated in ascending numeric sequence, thus making any portion of the LINKAGE SECTION easy to find in the program listing. Use of this convention allows the same data element to appear in many different locations without having to do the one thing that will most destroy a program's usability and maintainability -- making minor changes to the meaningful portion of the data name in order to make it unique, thus destroying the continuity of the program where that data element is concerned.

For the larger programs, with large amounts of storage required, the use of the dual method of "LSnn-" variables and "LS-" constants has numerous advantages. Those include:

1. Easy access to any portion of a long listing of storage through use of the numeric sequence.

2. Elimination of the problem of data-name uniqueness where the same data element occurs in multiple locations.

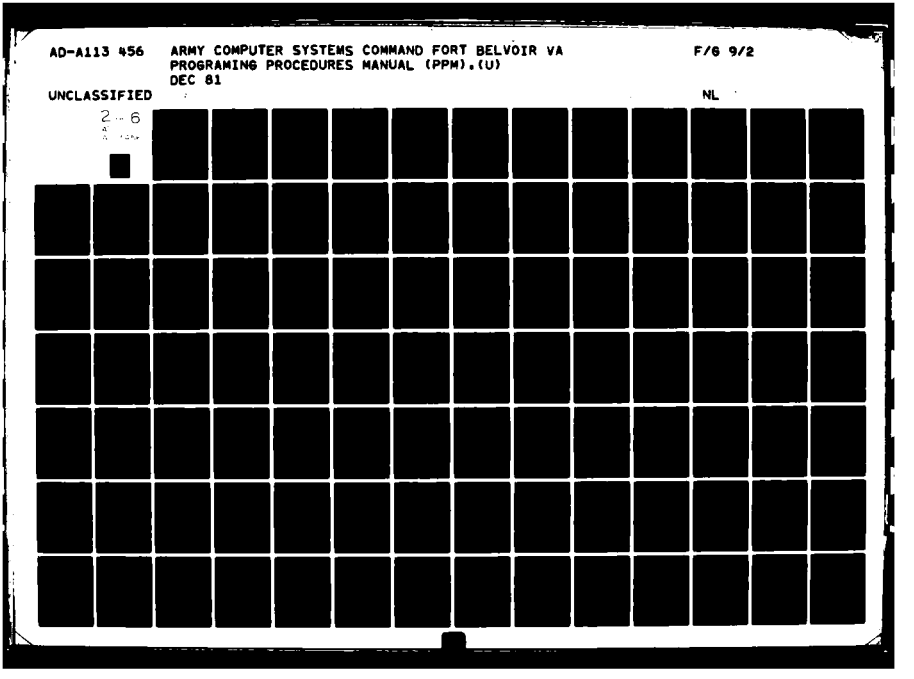
3. Unique, easily recognizable labeling of any area of storage through the designation of blocks of numbers to a given series of record types. For example, all report output record areas in the LINKAGE SECTION might be set up to carry prefixes in the 4Ø's; all input transaction record areas prefixes in the 5Ø's; etc.

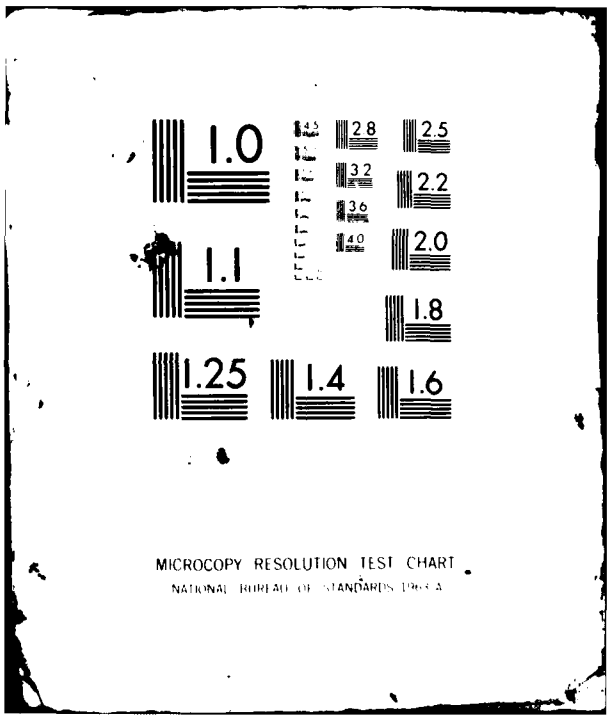
4. An extremely high number of areas may be set up using this method. Simply within the normal numeric series, up to 99 different blocks of data elements may be established.

5. For extremely large programs, though, or those with an incredibly high number of record description entries, the following extension to the "LSnn-" prefix is suggested. Once the entire 99 numbers have been employed, begin using a prefix of the structure "LSxn-" where the x is an alphabetic character beginning with the letter "A" and working up, with the n character, still a numeric digit, working through the sequence from zero (Ø) to nine (9) for each alphabetic character. Thus the first area would be prefixed "LSAØ-", the second "LSA1-", the third "LSA2-", etc. This method adds 26Ø more prefix combinations to the available set, certainly covering any foreseeable situations.

AD-A113 456 ARMY COMPUTER SYSTEMS COMMAND FORT BELVOIR VA F/G 9/2
PROGRAMING PROCEDURES MANUAL (PPM).(U)
DEC 81

UNCLASSIFIED NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

2.4.7.4 LINKAGE SECTION. (Cont.)

6. Using the dual method, the appearance of a "LS-" type prefix immediately labels that data-name to the programer as belonging to a constant whose value will not change. He doesn't have to constantly keep referring to his listing to determine which type of data element he is working with.

The "LSnn-" data-name prefixing applies to new programs under development. Existing programs will be converted to these standards mechanically through the use of USACSC COBOL formatting programs when and as convenient.

2.4.7.5 FILE DESCRIPTION (FD) AND SORT-FILE (SD) DESCRIPTION ENTRIES.

FUNCTION. The file description entry furnishes information concerning the physical structure, identification, and record-names necessary for processing a given external file. The sort-file description entry provides this information for a file that is to be sorted or merged.

FORMAT.

```
FD  file-name

  [; BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS    }]
                                             {CHARACTERS}

  [; RECORD CONTAINS     [integer-3 TO]    integer-4  CHARACTERS]

   ; LABEL  RECORDS ARE      {STANDARD}
                              {OMITTED }

  [; VALUE OF implementor-name-1 IS        {data-name-1}
                                           {literal-1  }

        [, implementor-name-2 IS    {data-name-2}      ] ...]
                                    {literal-2  }

  [; DATA {RECORD IS  } data-name-3 [,data-name-4]  ...  ]
          {RECORDS ARE}
   [CODE-SET IS alphabet-name.]

SD  file-name

  [; RECORD CONTAINS [integer-1 TO] integer-2  CHARACTERS]

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

2.4.7.5 FILE DESCRIPTION (FD) AND SORT-FILE (SD) DESCRIPTION ENTRIES. (Cont.)

SYNTAX RULES.

- The level indicator (FD or SD) starts in the first position of Margin 'A' and signifies the beginning of the file (or sort) description.
- The file-name immediately follows the level indicator.
- The file and/or sort descriptions are only allowed in the FILE SECTION of the DATA DIVISION.

GENERAL RULES.

- The file description entry must have one or more record description entries subordinate to it. If it has more than one, internal storage is reserved for only the largest record description. Each additional record description is assumed to redefine the largest one and occupies the same storage area. Redefinition is implied.
- The file description entry for file-name must be equivalent to that used when this file was created.

USACSC GUIDELINES.

- For ease of maintenance of modified record format, it is suggested that all record description entries be kept on a direct-access storage device and called into the program using the COPY verb.
- Each clause under the FD or SD entry must be on a separate line.

2.4.7.6 LABEL RECORDS CLAUSE.

FUNCTION. The LABEL clause specifies the presence of labels for a file.

FORMAT.

```
LABEL    RECORDS ARE  { STANDARD }
                      { OMITTED  }
```

SYNTAX RULES. The LABEL RECORDS clause is required in each file description.

GENERAL RULES. STANDARD signifies the existence of file labels which conform to system specification.

USACSC GUIDELINES. All output files must have standard labels where file processing conditions do not prohibit them.

2.4.7.7 RECORD CONTAINS CLAUSE.

FUNCTION. The RECORD CONTAINS clause specifies the size of data records.

FORMAT.

```
[RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]
```

SYNTAX RULES. Integer-3 and integer-4 must be positive, unsigned integers.

GENERAL RULES.

- In a variable length file, integer-3 refers to the minimum record length and integer-4 refers to the maximum record length.
- Integer-4 may never be zero and should be used alone only for files containing fixed length records.
- Integer-3 and integer-4 are expressed in terms of actual bytes of data required for the record.
- Actual record lengths are computed by the compiler from the record descriptions.

VENDORS' GUIDELINES.

- IBM.

If the record description for a file contains more than one OCCURS DEPENDING ON clause, the maximum compiler-calculated size may be greater than needed. The user may override this calculation by specifying the size desired in integer-4. In this case, the user-specified value of integer-4 determines the amount of storage set aside to contain the data record.

USACSC GUIDELINES. The use of this clause is recommended.

2.4.7.8 VALUE OF CLAUSE.

FUNCTION. The VALUE OF clause particularizes the description of an item in the label records associated with a file.

2.4.7.8 VALUE OF CLAUSE. (Cont.)

FORMAT.

```
[  VALUE OF implementor-name-1 IS { data-name-1 }                    ]
[                                 { literal-1   }                    ]
[       [ , implementor-name-2 IS { data-name-2 } ] ...              ]
[                                 { literal-2   }                    ]
```

SYNTAX RULES.

- Data-name-1, data-name-2, etc., should be qualified when necessary, but cannot be subscripted or indexed, nor can they be items described with the USAGE IS INDEX clause.

- Data-name-1, data-name-2, etc., must be in the Working-Storage Section.

GENERAL RULES.

- For an input file, the appropriate label routine checks to see if the value of implementor-name-1 is equal to the value of literal-1 or data-name-1.

- For an output file, at the appropriate time the value of implementor-name-1 is made equal to the value of literal-1 or data-name-1.

- A figurative constant may be substituted in the format above wherever a literal is specified.

VENDORS' GUIDELINES.

- IBM.

  In both OS and DOS, this clause is treated as a comment.

USACSC GUIDELINES. None.

2.4.7.9 BLOCK CONTAINS CLAUSE.

FUNCTION. The BLOCK CONTAINS clause specifies the size of the physical records (blocks) which contain the logical records in a file.

FORMAT.

```
[  BLOCK CONTAINS [integer-1 TO] integer-2 { CHARACTERS }]
[                                          { RECORDS    }]
```

2.4.7.9 BLOCK CONTAINS CLAUSE. (Cont.)

SYNTAX RULES. Integer-1 must be a positive unsigned integer.

GENERAL RULES.

- If this clause is not used, a physical record containing one and only one complete logical record is assumed.

- Use of the RECORDS option: This option states the number of logical records to be placed in a physical block. The compiler computes the block size to be the sum of integer-2 records of maximum size plus any required control bytes.

- Use of CHARACTERS option:

If neither RECORDS nor CHARACTERS are specified, CHARACTERS is assumed.

The CHARACTERS option represents the exact size of the physical record in terms of the number of character positions required to store the physical record.

Techniques for handling physical records (blocks) containing variable-length logical records are specified by each vendor.

- If integer-1 and integer-2 are both shown, they refer to the minimum and maximum size of the physical record, respectively.

VENDORS' GUIDELINES.

- IBM.

IBM specifies that for variable-length files (mode of V) a block descriptor field is appended to the beginning of each block. This is a four byte field which must be added to the desired actual block size (integer-2) when the CHARACTERS option is used.

The CHARACTERS option represents the exact size of the physical record in terms of the number of bytes occupied internally by its logical records plus slack bytes.

The CHARACTERS clause need not be specified for U-mode files.

An IBM OS extension allows that integer-2 be set to zero. If zero is specified, the block size is determined at object time from the DD parameters or the data set label for the file. The file name may not appear in a SAME AREA clause and the file must either be a sequential or indexed file whose ACCESS MODE is sequential.

2.4.7.9 BLOCK CONTAINS CLAUSE. (Cont.)

USACSC GUIDELINES.

- IBM.

The following is applicable for IBM-OS COBOL:

- BLOCK CONTAINS Ø clause is recommended for all sequential input files, tapes, or DASD sequential output files.

- BLOCK CONTAINS Ø option should not be used with variable-length records since it may lead to inefficient blocking of logical records. Also, index-sequential files cannot use block contains zero clause.

- BLOCK CONTAINS Ø option cannot be used for those files utilizing the SAME AREA clause.

2.4.7.10 CODE-SET CLAUSE.

FUNCTION. The CODE-SET clause specifies the character code set used to represent data on the external media.

FORMAT.

```
[ CODE-SET IS alphabet-name ]
```

SYNTAX RULES.

- When the CODE-SET clause is specified for a file, all data in that file must be described as usage is DISPLAY and any signed numeric data must be described with the SIGN IS SEPARATE clause.

- The alphabet-name clause referenced by the CODE-SET clause must not specify the literal phrase.

- The CODE-SET clause may only be specified for non-mass storage files.

GENERAL RULES.

- If the CODE-SET clause is specified, alphabet-name specifies the character code convention used to represent data on the external media. It also specifies the algorithm for converting the character codes on the external media from/to the native character codes. This code conversion occurs during the execution of an input or output operation. (See paragraph 2.4.6.5, SPECIAL-NAMES PARAGRAPH).

- If the CODE-SET clause is not specified, the native character code set is assumed for data on the external media.

USACSC GUIDELINES. None.

2.4.7.11 DATA RECORD CLAUSE.

FUNCTION. The DATA RECORDS clause merely documents the names of the records in a file.

FORMAT.

```
[ DATA { RECORD IS  } data-name-3 [,data-name-4] ... ]
       { RECORDS ARE}
```

SYNTAX RULES. Data-name-3 and data-name-4 are the names of data records in the file. They are the names of the Ø1-level record description entries for that file.

GENERAL RULES.

- The use of more than one data-name indicates the presence of more than one type of data record. These data records may have different descriptions.
- Multiple data records for a file share the same storage area.
- The order in which data-names are listed is unimportant.

USACSC GUIDELINES. None.

2.4.7.12 RECORD DESCRIPTION CLAUSE.

FUNCTION. The RECORD DESCRIPTION entry consists of the various levels of data description entries which describe the record(s) contained in a file.

FORMAT.

```
level-number  data-name  COPY  library-name
```

SYNTAX RULES.

- A level-number of Ø1 signifies the beginning of the record description.
- The record description is terminated by the next level-number of Ø1 or the next level indicator of SD or FD.

2.4.7.12 RECORD DESCRIPTION CLAUSE. (Cont.)

GENERAL RULES.

- Every file description (FD) entry or sort description (SD) entry must have at least one record description describing it.
- The record description entry is valid in all sections of a COBOL source program.
- If the record description is not subdivided, it is also a data description entry and must follow the rules for data description entry development.
- The record description may be copied from a direct-access storage library.

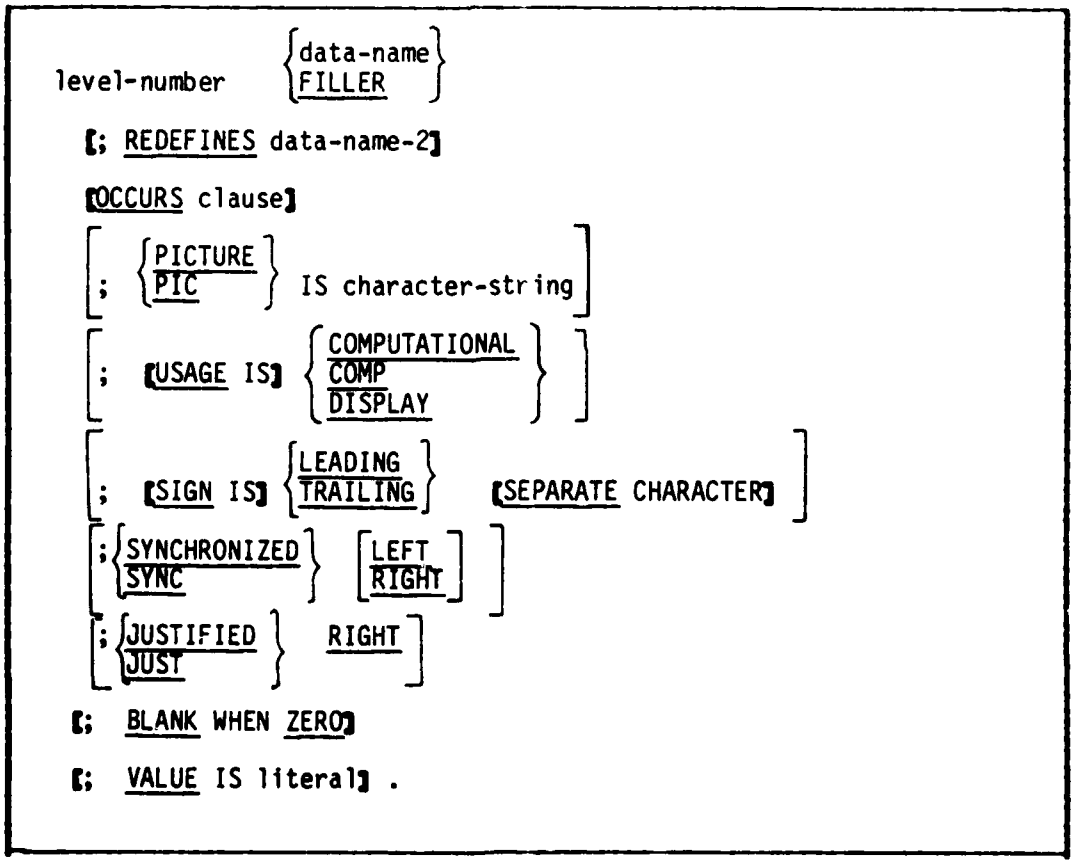
USACSC GUIDELINES. It is suggested that for ease of maintenance of record formats, that the record description be copied from the direct-access storage library.

2.4.7.13 DATA DESCRIPTION CLAUSE.

FUNCTION. The DATA DESCRIPTION entry describes the characteristics of a 77-level data item, an 88-level condition name, and all elementary and group data items subordinate to a record description. It consists of a level-number and data-name, plus any applicable data description clauses.

FORMAT.

FORMAT 1.



2.4.7.13 DATA DESCRIPTION CLAUSE. (Cont.)

FORMAT 2.

| |
|---|
| 88 condition-name VALUE clause |

SYNTAX RULES.

- An entry plus its clauses must be terminated by a period.
- The maximum length for a data description entry is vendor specified.
- The PICTURE clause must be specified for every elementary item except an index data item, in which case use of this clause is prohibited.
- Condition-name 88-level entries must immediately follow the data description entry of which they are a variable.

GENERAL RULES.

- Data description clauses and 77-level data items are to be written in the order outlined in FORMAT 1. However, unnecessary clauses may be omitted.
- The PICTURE, JUSTIFIED and BLANK WHEN ZERO clauses may only be specified for elementary items.
- The following rules apply to the use of 88-level condition names:

A condition-name can be associated with any elementary item except another condition-name or index data item.

A condition-name can be associated with a group item with the following restrictions:

1. The value must be a non-numeric literal or figurative constant.
2. The size of the condition cannot exceed the total of the sizes of the elementary items in the group.
3. No element in the group may contain a JUSTIFIED clause.
4. All items within the group must have a USAGE of DISPLAY.
5. Condition-names may also be applied to levels subordinate to the group item that has a condition-name associated with it.

2.4.7.13 DATA DESCRIPTION CLAUSE. (Cont.)

VENDORS' GUIDELINES.

- IBM.

*The maximum length for a data description entry is 32,767 bytes except* for a fixed-length WORKING-STORAGE or LINKAGE SECTION group item which may be 131,Ø71 bytes.

USACSC GUIDELINES. When used, data description clauses should be arranged in the following order:

| | |
|---|---|
| REDEFINES | VALUE |
| OCCURS | JUSTIFIED |
| PICTURE | SYNCHRONIZED |
| USAGE | BLANK WHEN ZERO |

2.4.7.14 DATA-NAME OR FILLER CLAUSE.

FUNCTION. A data-name refers to the name of the data item being described. The word FILLER refers to an elementary item of a logical record which is never referenced and need not be named.

FORMAT.

```
level-number { data-name }
             { FILLER    }
```

SYNTAX RULES. Data-name or FILLER must immediately follow the level-number.

GENERAL RULES.

- The key word FILLER may never be directly referenced.
- In the WORKING-STORAGE and LINKAGE SECTIONS, level-77 and level-Ø1 entries should be given unique data-names.
- A data-name subordinate to another data-name must be unique.
- A data-name refers to a type of data, not a specific value. The value of a data-name may vary throughout the program.

2.4.7.14 DATA-NAME OR FILLER CLAUSE. (Cont.)

• The key word FILLER is to be used only wi h the elementary level items. Group level items will have unique descriptive labels.

USACSC GUIDELINES.

• When the term FILLER is used it will be spelled in full.

• The WORKING-STORAGE and LINKAGE SECTIONS will be prefixed by "WS-" or "WSnn-" and "LS" or "LSnn", respectively. Reference WORKING-STORAGE SECTION and LINKAGE SECTION.

• Data-names under each file description will follow the rules prescribed in the FILE SECTION of the DATA DIVISION.

2.4.7.15 REDEFINES CLAUSE.

FUNCTION. The REDEFINES clause provides a means for giving different definitions to the same area of computer storage. This implies a redefinition of the physical storage area not the data items describing the area. The redefinition may involve structure usage, or rearrangement of the storage positions.

FORMAT.

```
level-number    data-name-1    REDEFINES    data-name-2
```

NOTE: Level-number and data-name-1 are shown in the above format to improve clarity; they are not part of the REDEFINES clause.

SYNTAX RULES.

• The REDEFINES clause, when specified, must immediately follow data-name-1.

• The level-numbers of data-name-1 and data-name-2 must be the same but must not be an 88-level.

• This clause may not be used at the Ø1-level in the FILE SECTION. When more than one Ø1-level entry applies to a file description, redefinition is implied.

GENERAL RULES.

• The area to be redefined starts at data-name-2, includes all data items subordinate to data-name-2, and ends when a level-number equal to or less than

2.4.7.15 REDEFINES CLAUSE. (Cont.)

data-name-2 is reached. This point should be the data description entry for data-name-1. There can be no intervening entries with numerically lower level-numbers than data-name-1 and data-name-2.

- The computed length of data-name-1 must equal the computed length of data-name-2.

- The USAGE may be changed by the redefinition. This does not, however, change the existing data.

- The entries making up the new definition (data-name-1 and its subordinate entries) may not contain VALUE clauses.

- Multiple redefinitions of the same area are allowed. However, data-name-2 in all redefinitions will be the same as the data-name which originally described the area.

- Redefinitions may be nested. A redefining entry may have another redefining entry subordinate to it.

- Certain restrictions are necessary when using the OCCURS clause in conjunction with the REDEFINES.

  The area being redefined (data-name-2) cannot contain an OCCURS clause.

  Data-name-2 cannot be subordinate to an entry containing an OCCURS clause.

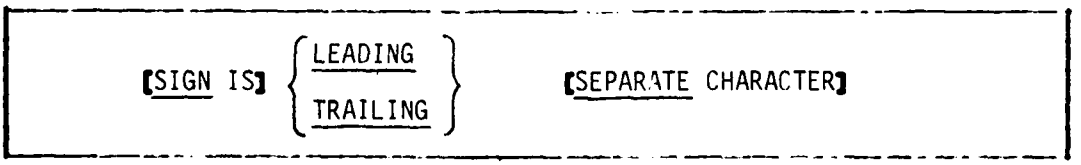
  Data-name-1 and its subordinates and subordinates to data-name-2 may contain the OCCURS clause without the DEPENDING ON option.

USACSC GUIDELINES. None.

2.4.7.16 SIGN CLAUSE.

FUNCTION. The SIGN clause specifies the position and the mode of representation of the operational sign when it is necessary to describe these properties explicitly.

FORMAT.



2.4.7.16 SIGN CLAUSE. (Cont.)

SYNTAX RULES.

• The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character 'S', or a group item containing at least one such numeric data description entry.

• The numeric data description entries to which the SIGN clause applies must be described as usage is DISPLAY.

• At most one SIGN clause may apply to any given numeric data description entry.

• If the CODE-SET clause is specified, any signed numeric data description entries associated with that file description entry must be described with the SIGN IS SEPARATE clause.

GENERAL RULES.

• The optional SIGN clause, if present, specifies the position and the mode of representation of the operational sign for the numeric data description entry to which it applies, or for each numeric data description entry subordinate to the group to which it applies. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character 'S'; the 'S' indicates the presence of, but neither the representation nor, necessarily, the position of the operational sign.

• A numeric data description entry whose PICTURE contains the character 'S', but to which no optional SIGN clause applies, has an operational sign, but neither the representation nor, necessarily, the position of the operational sign is specified by the character 'S'. In this (default) case, the implementor will define the position and representation of the operational sign. The following general rules do not apply to such signed numeric data items.

• If the optional SEPARATE CHARACTER phrase is not present, then:

a. The operational sign will be presumed to be associated with the leading (or, respectively, trailing) digit position of the elementary numeric data item

b. The letter 'S' in a PICTURE character-string is not counted in determining the size of the item (in terms of standard data format characters).

c. The implementor defines what constitutes valid sign(s) for data items.

• If the optional SEPARATE CHARACTER phrase is present, then:

a. The operational sign will be presumed to be the leading (or, respectively, trailing) character position of the elementary numeric data item; this character position is not a digit position.

b. The letter 'S' in a PICTURE character-string is counted in determining the size of the item (in terms of standard data format characters).

2.4.7.16 SIGN CLAUSE. (Cont.)

c. The operational signs for positive and negative are the standard data format characters '+' and '-', respectively.

• Every numeric data description entry whose PICTURE contains the character 'S' is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

USACSC GUIDELINES. None.

2.4.7.17 OCCURS CLAUSE.

FUNCTION. The OCCURS clause eliminates the need for separate entries for repeated data by indicating the number of times a series of data items with identical format is repeated. It defines tables and supplies information on the application of subscripts and indexes.

FORMAT.

FORMAT 1.

```
OCCURS       integer-2  TIMES

[ { ASCENDING  } KEY IS data-name-2 [, data-name-3] ... ] ...
  { DESCENDING }

[ INDEXED BY  index-name-1  [, index-name-2] ...]
```

FORMAT 2.

```
OCCURS     integer-1 TO integer-2 DEPENDING ON data-name-1

[ { ASCENDING  } KEY IS data-name-2 [, data-name-3] ...  ] ...
  { DESCENDING }

  INDEXED BY  index-name-1 [, index-name-2] ...]
```

2.4.7.17 OCCURS CLAUSE. (Cont.)

SYNTAX RULES

- Integer-1 and integer-2 must be positive integers. Where both are used the value of integer-2 must be greater than value of integer-1.

- The OCCURS clause cannot be specified for an entry that has a 77, 88 or Ø1-level number.

- The OCCURS clause may not have a variable size item subordinate to it. That is, an OCCURS clause with the DEPENDING ON option cannot be contained within another OCCURS clause.

- A VALUE clause may not be used in a description containing an OCCURS clause or in a description subordinate to it.

- Data-name-1 must describe a positive integer.

- Data-name-1, data-name-2, ... may be qualified.

- Data-name-2 must either be the name of the entry containing the OCCURS clause or the name of an entry subordinate to the entry containing the OCCURS clause.

- Data-name-3, etc., must be the name of an entry subordinate to the group item which is the subject of this entry.

- An entry which contains an OCCURS DEPENDING ON clause or which has a subordinate entry containing this clause cannot be redefined.

- These clauses must be in the order shown.

GENERAL RULES.

- Any clauses used in conjunction with an OCCURS clause applies to each occurrence of the data item.

- Three levels of subscripting or indexing are allowed, therefore, only three nested levels of OCCURS clauses are allowed.

- The subject of an OCCURS clause is the data-name of the entry containing the clause. Whenever the subject or a data item subordinate to it are referenced (except in the SEARCH statement), it must be subscripted or indexed.

- Rules for integer-1 and integer-2 with the OCCURS clause are:

  In FORMAT 1, integer-2 represents the exact number of occurrences of the subject of the occurs. It must be greater than zero.

  In FORMAT 2, integer-1 states the minimum number of occurrences of the subject of the occurs and may be zero or greater.

2.4.7.17 OCCURS CLAUSE. (Cont.)

In FORMAT 2, integer-2 states the maximum number of occurrences and must be greater than zero. The value of data-name-1 cannot be greater than integer-2.

- The DEPENDING ON option indicates that the subject of the entry may occur a varying number of times. The value in data-name-1 determines the number of occurrences. Data-name-1 has the following restrictions:

  It must be a positive integer equal or less than integer-2.

  It may not be part of a table and, therefore, may not be subscripted.

  If data-name-1 is contained in the same record as the variable length table data, data-name-1 must be in a fixed portion of the record. It cannot be in the variable length portion.

  If the value of data-name-1 is reduced, the contents of data items exceeding the new number of occurrences in data-name-1 is unpredictable.

- The KEY option is used when the table is arranged in sequence and the SEARCH ALL statement is to be used. The KEY option states the sequence of the repeated fields, ASCENDING or DESCENDING, according to the values of data-name-2, data-name-3, etc. The data-names have certain restrictions.

  If data-name-2 is the name of the entry containing the OCCURS clause, it is the only key that may be specified.

  If data-name-2 is not the name of the entry containing the OCCURS clause, it must be an entry subordinate to it. In this case, multiple keys may be used with the following rules.

  1. The keys may be DISPLAY or COMP.
  2. All keys must be subordinate to another entry containing an OCCURS clause.
  3. The keys may not contain an OCCURS clause.
  4. The keys must be listed in order of significance.

- The INDEXED BY option is used if the subject of the entry or an item subordinate to it are to be referred to by indexing. The index-name is not defined elsewhere in the program since its allocation and format are dependent on the system. It is not data and, therefore, it cannot be associated with a data hierarchy.

  No more than 12 index-names may be used per entry.

  The index-name must be initialized by a SET statement before use.

2.4.7.17 OCCURS CLAUSE. (Cont.)

An index-name is a fullword and represents a binary value of the actual displacement from the beginning of the table to the occurrence number in the table. This is calculated in the following way:

```
(occurrence number minus 1)  *  (length of entry)
```

If the entry is set up this way,

```
A OCCURS 15 TIMES INDEXED BY N PIC X(10).
```

on the fifth occurrence of A,

```
INDEX-NAME = (5-1)  *  10 = 40.
```

- The KEY option and INDEXED BY option are further developed in paragraph 2.5.4, TABLE HANDLING FEATURE.

VENDORS' GUIDELINES.

- IBM.

  In FORMAT 2, integer-2 must be less than 32,768 bytes.

  Using the KEY option, the total number of keys per table element is 12. The sum of the lengths of all the keys for a particular table element may not be more than 256.

USACSC GUIDELINES. None.

2.4.7.18 PICTURE CLAUSE.

FUNCTION. The PICTURE clause describes the general characteristics and editing requirements of an elementary item.

FORMAT.

```
{PICTURE}
{PIC    }    IS character-string
```

SYNTAX RULES.

- PIC is an abbreviation of PICTURE and is the preferred form.
- A PICTURE clause may only be specified at the elementary item level.
- The PICTURE clause must be specified for each elementary item except index data items for which no PICTURE clause is allowed.
- The character-string must contain only certain allowable combinations of symbols from the COBOL character set. The allowable combinations determine the category of the elementary item.
- The maximum number of positions coded for a character-string is 30.
- The asterisk when used as the zero suppression symbol and the clause BLANK WHEN ZERO may not appear in the same entry.

GENERAL RULES.

- For elementary items data is classified into five categories: Alphabetic, numeric, alphanumeric, alphanumeric-edited and numeric-edited.
- Group items are always considered to be in the alphanumeric class.
- The size of an elementary item is a function of the character positions in the PICTURE character-string and the USAGE clause.
- The PICTURE clause may be written with a symbol and a following integer inclosed in parentheses. The combination X(10) refers to a character-string of 10 X's.
- The following symbols may appear only once in a PICTURE clause.

```
S    V    .    CR    DB
```

2.4.7.18 PICTURE CLAUSE. (Cont.)

- The symbols used in a PICTURE clause are:

A The 'A' represents a character position which can only be filled by a letter of the alphabet or a space.

B The 'B' represents a character position where a space character will be inserted.

P The 'P' represents an assumed decimal scaling position. It is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The 'P' is not counted in the size of the data item but is counted toward determining the maximum number of digit positions allowable (18) in numeric-edited items or in items that appear as arithmetic operands. The scaling character 'P' may only appear in a continuous string (may not be intermixed with other symbols) to the left or right of other characters in the picture. The sign character 'S' and the assumed decimal point 'V' are the only symbols which may appear to the left of a leftmost string of 'P's. Since the 'P' implies an assumed decimal point (to the left of the string of 'P's if they are the leftmost characters and to the right if they are the rightmost characters) the assumed decimal point V is redundant.

S The 'S' indicates the presence of an operational sign (but not its position or representation). It is written in the leftmost position of the character-string. The 'S' is not counted in the size of the item unless a SIGN clause with the SEPARATE CHARACTER option is associated with it. May only appear once per *PICTURE* clause.

V The 'V' indicates the position of an assumed decimal point and may appear only once in a character string. The 'V' does not represent a character position; it is used for alignment only, and therefore not counted in the size of the item. When the 'V' is the rightmost symbol in the string, it is redundant.

X An 'X' represents a character position which may contain any character from the COBOL character set.

Z A 'Z' in the character string represents a leading numeric character position. Its purpose is for editing. Whenever the character position it represents contains a zero, a space is substituted for the zero. The 'Z' is counted in the size of the item.

9 The '9' represents a character position that can contain only a numeral. The '9' is counted in the item size.

Ø A zero represents a position where the numeral zero will be inserted. Each zero is counted in the size of the item.

2.4.7.18 PICTURE CLAUSE. (Cont.)

/ Each '/' (stroke) in the character-string represents a character position into which the stroke character will be inserted. The '/' is counted in the size of the item.

, The comma represents a position where a comma will be inserted. The comma cannot be the last character in the picture. It is counted in the size of the item.

. A period represents the decimal point for alignment purposes. It is also an editing symbol which inserts a period in that position. It cannot be the last character in a string and is counted in the size of the item.

+ - CR DB These symbols are used as editing sign control symbols. Each represents the position into which the editing sign control symbol will be placed. Only one type of these symbols may be used per character string. These are counted in determining the size of the data item. A signed numeric literal cannot be used in a value clause unless it is associated with a signed PICTURE character-string.

* An asterisk represents a leading numeric character position into which an asterisk will be placed when that position contains a zero. The asterisks are counted in the size of the item. When representing a data item with an asterisk in its PICTURE clause, the BLANK WHEN ZERO clause cannot be used in that entry.

$ The currency symbol represents a position into which a dollar sign is to be placed. This symbol is counted in the size of the item.

- The five categories of data and their characteristics are:

An alphabetic item's picture contains only the symbol 'A'. Its value must be a combination of the 26 letters of the Roman alphabet or a space. Each character is stored in a separate position. If the VALUE clause is specified, the literal must be non-numeric.

The picture for a numeric item is restricted to symbols '9', 'V', 'P', and 'S'. The maximum number of digit positions permissible is 18. The contents must be Arabic numerals from Ø through 9 and it may contain an operational sign. If the PICTURE contains an 'S', the contents of the item are positive or negative depending on the sign. If the PICTURE has no 'S', the contents of the item is considered an absolute value. A value specified for an elementary numeric item must be numeric.

2.4.7.18 PICTURE CLAUSE. (Cont.)

An alphanumeric item has a PICTURE character-string which contains combinations of 'A', 'X', and '9'. The item is treated as if the character-string was all 'X's. A character-string of all 'A's or all '9's does not constitute an alphanumeric. Its contents must be from the Vendor's character set. All group items are treated as alphanumerics and any values assigned must be non-numeric.

An alphanumeric edited item is restricted to combinations of the 'A', 'X', '9', 'B', Ø and '/' symbols. The character-string must contain at least one 'A' or 'X', and must contain at least one '/', 'B' or 'Ø'. The contents must be characters from the Vendor's character set. If a value is specified, it must be non-numeric. No editing will be done on the value assigned; it is treated exactly as stated.

A numeric-edited item is restricted to the use of the following symbols:

| B / P V Z Ø 9 , . * + - CR DB $ |
|---|

The literal in the VALUE clause of numeric-edited data items must be non-numeric.

The maximum number of digit positions must not exceed 18 and the contents of character positions which represent digits must be numerics. The character-string must contain at least one of the following: 'Ø', 'B', '/', 'Z', '*', ',', '.', '-', 'CR', 'DB' or the currency symbol. Since a numeric-edited item is DISPLAY, a value specified must be non-numeric. The literal is placed in the value as stated, no editing takes place for initial values.

- Editing is accomplished in the PICTURE clause using two methods: insertion editing or suppression and replacements. There are four types of insertion editing: simple insertion, special insertion, fixed insertion, and floating insertion. There are two types of suppression and replacement editing: zero suppression and replacement with spaces and zero suppression and replacement with asterisks.

- Floating insertion editing and editing by zero suppression and replacement may not be used with any other forms of editing. Only one type of replacement (space or asterisk) may be used with zero suppression in the PICTURE clause.

- Simple insertion editing uses only the ',' (comma), 'B' (space) and 'Ø' (zero) as insertion characters. These characters are counted in the size of the item and represent the position where the comma, space, or zero is to be inserted. See FIGURE 2-8.

2.4.7.18 PICTURE CLAUSE. (Cont.)

| PICTURE | VALUE | RESULT |
|---|---|---|
| 99,999 | 12345 | 12,345 |
| 9,999,000 | 12345 | 2,345,000 |
| 99B999B000 | 1234 | 01 234 000 |
| 99B999B000 | 12345 | 12 345 000 |
| 99BBB999 | 123456 | 23 456 |

FIGURE 2-8

• Special insertion editing uses the '.' (period) as the insertion character. It also represents the decimal point for alignment purposes and as such is counted in the size of the item. The use of the assumed decimal point (V) and the actual decimal point (.) is not allowed in the same picture clause. The result of this type of editing is the placement of the period in the same position as shown in the character-string. See FIGURE 2-9.

| PICTURE | VALUE | RESULT |
|---|---|---|
| 999.99 | 1.234 | 001.23 |
| 999.99 | 12.34 | 012.34 |
| 999.99 | 123.45 | 123.45 |
| 999.99 | 1234.5 | 234.50 |

FIGURE 2-9

• Fixed insertion editing uses the currency symbol ($) and the editing sign control symbols '+', '-', 'CR' and 'DB' as insertion characters. Only one currency symbol and one sign control symbol may appear in one PICTURE character-string. The insertion character will occupy the same position in the edited item as it occupied in the PICTURE character-string. Restrictions on the use of these symbols are:

$ — The currency symbol may be preceded by '+' or '-'; otherwise, it must be the leftmost character in the character-string. It is counted in the size of the item.

+ - — One of these symbols ('+' or '-') must be the rightmost or leftmost character position and is counted in the size of the item.

CR DB — These symbols ('CR' or 'DB') represent two rightmost character positions and are counted in the size of the item. See FIGURE 2-10.

2.4.7.18 PICTURE CLAUSE. (Cont.)

| PICTURE | VALUE | RESULT |
|---|---|---|
| 999.99+ | +1234.567 | 234.56+ |
| +9999.99 | -1234.567 | -1234.56 |
| 9999.99- | +1234.56 | 1234.56 |
| $999.99 | -123.45 | $123.45 |
| -$999.99 | -123.45 | -$123.45 |
| $9999.99CR | +123.45 | $0123.45 |
| $9999.99DB | -123.45 | $0123.45DB |

FIGURE 2-10

• For floating insertion editing, the currency symbol ($) and editing sign symbols (+ or -) are used as insertion characters and are mutually exclusive in a PICTURE character-string. At least two of the floating insertion characters must appear in a string as the leftmost character. Fixed insertion symbols may be embedded in the floating character-string or may be to the right of the string. In either case, they are counted as part of the floating character-string.

There are two ways of representing floating insertion editing in a PICTURE character-string. Each way causes a different result in the edited field.

The first way is to represent any or all of the leading numeric character positions to the left of the decimal point by insertion characters. This will result in an edited field with a single insertion character placed in the character position immediately preceding the first non-zero digit in the data or the decimal point, whichever is leftmost. The character positions in the floating character-string that are left of the inserted symbol are replaced with spaces.

The second way is to represent all the numeric character positions in the PICTURE character-string by the insertion character. The edited results of this format is dependent on the value of the data. If the data is equal to zero, the entire edited field will contain spaces. If the value is not zero, the same type of editing applies as above. A single insertion character will be placed before the last non-zero digit or the decimal point (whichever is leftmost) and the remaining insertion symbols to the left will be replaced by spaces.

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of nonfloating insertion characters being edited into the receiving data item, plus one for the floating insertion character. See FIGURE 2-11.

2.4.7.18 PICTURE CLAUSE. (Cont.)

| PICTURE | VALUE | RESULT |
|---|---|---|
| $$$$.99 | .123 | $.12 |
| $$$9.99 | .12 | $0.12 |
| $$,$$$,999.99 | -1234.56 | $1,234.56 |
| ++,+++,999.99 | -123456.789 | -123,456.78 |
| $$,$$$,$$$.99CR | -1234567 | $1,234,567.00CR |
| $$,$$$,$$$.99DB | +1234567 | $1,234,567.00 |
| ++,+++,+++.+++ | 00C0.00 | |

FIGURE 2-11

• The zero suppression and replacement form of editing means the suppression of leading zeroes in a numeric item and replacing them with spaces or asterisks. If the alphabetic character 'Z' is used as the insertion symbol, the replacement character is space. If the insertion symbol is '*', the replacement character will be '*'. The 'Z' and '*' may not be mixed in the same PICTURE. Each suppression symbol is counted in the size of the item.

Any simple insertion characters embedded in the string or to the right of the string are part of the string. Simple insertion or fixed insertion editing characters to the left of the string are not included.

There are two ways of representing zero suppression in a PICTURE character-string. Any or all of the characters left of the decimal point may be represented by suppression symbols; or all of the characters may be represented by suppression symbols. In the first instance, leading zeroes are replaced with replacement characters from the left until the first non-zero digit or the decimal point is encountered. In the second instance, all leading zeroes will be replaced regardless of the decimal point - if the value is zero, the entire data item will be spaces or asterisks except for the decimal point. See FIGURE 2-12.

| PICTURE | VALUE | RESULT |
|---|---|---|
| ZZZZ.ZZ | 0000.00 | |
| ****.** | 0000.00 | ****.** |
| ZZZZ.99 | 0000.00 | .00 |
| ****.99 | 0000.00 | ****.00 |
| ZZ99.99 | 0000.00 | 00.00 |
| Z,ZZZ.ZZ+ | +123.456 | 123.45+ |
| *,***.**+ | -123.456 | **123.45- |
| $Z,ZZZ,ZZZ.ZZCR | +12345.67 | $12,345.67 |
| $B*,***,***.**BBDB | -12345.67 | $***12,345.67 DB |

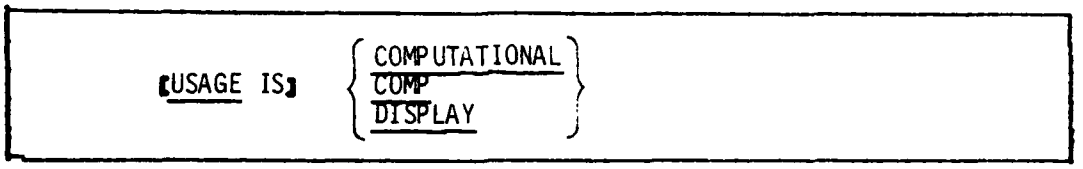
FIGURE 2-12

VENDORS' GUIDELINES. The character set for IBM is the EBCDIC character set.

2.4.7.19 USAGE CLAUSE.

FUNCTION. The USAGE clause specifies the format of a data item in the computer storage.

FORMAT.




SYNTAX RULES.

- The PICTURE character-string of a COMPUTATIONAL item can contain only '9's, the operational sign character 'S', the implied decimal point character 'V', one or more 'P's.

- COMP is an abbreviation for COMPUTATIONAL.

GENERAL RULES.

- The USAGE clause can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

- This clause specifies the manner in which a data item is represented in the storage of a computer. It does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referred to. The USAGE clause may affect the radix or type of character representation of the item.

- A COMPUTATIONAL item is capable of representing a value to be used in computations and must be numeric. If a group item is described as COMPUTATIONAL, the elementary items in the group are COMPUTATIONAL. The group item itself is not COMPUTATIONAL (cannot be used in computations).

- The USAGE IS DISPLAY clause indicates that the format of the data is a standard data format.

- If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, the usage is implicitly DISPLAY.

USACSC GUIDELINES. None.

2.4.7.20 VALUE CLAUSE.

FUNCTION. The VALUE clause is used to define the initial value of a WORKING-STORAGE item.

FORMAT.

FORMAT 1.

```
VALUE IS literal
```

FORMAT 2.

```
{VALUE IS  } literal-1 [{THROUGH} literal-2 ]
{VALUES ARE}           {THRU   }

[ ,literal-3 [{THROUGH} literal-4 ] ]...
             {THRU   }
```

SYNTAX RULES.

- The VALUE clause may not be used for a variable length item.
- A figurative constant may be used instead of a literal.

GENERAL RULES.

- The VALUE clause is only applicable to the WORKING-STORAGE SECTION. VALUE can be specified in LINKAGE SECTION or FILE SECTION only with condition-names.
- The VALUE clause specifies the initial value of an item.

2.4.7.20 VALUE CLAUSE. (Cont.)

- Certain rules pertain to usage of the VALUE clause:

If the item is numeric, the literals must be numeric. The literal will be aligned from right to left and must not require truncation. In other words, the value must be within the numeric range described by the PICTURE clause.

If the item is alphabetic or alphanumeric, the literals must be non-numeric. The literal will be aligned from left to right and must not require truncation.

Editing characters in the PICTURE clause are used to determine the size of the item. Therefore, the value for an edited item must appear in the edited form.

- The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item.

- The VALUE clause cannot be specified in a data description entry which contains either an OCCURS clause or a REDEFINES clause. It also cannot be specified in a data description entry which is subordinate to an entry containing an OCCURS or REDEFINES clause.

- If a VALUE clause is used with a group level item, the literal must be a figurative constant or a non-numeric literal. The group is initialized as though the USAGE was display and no consideration is made of the USAGE of the subordinate items within the group. Subordinate levels of entries may not contain a VALUE clause.

- The VALUE clause cannot be specified for a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

- FORMAT 2 can be specified only for a condition-name (level 88) entry. Literal-2 must be larger than literal-1, and literal-4 must be larger than literal-3, etc.

VENDORS' GUIDELINES.

- IBM.

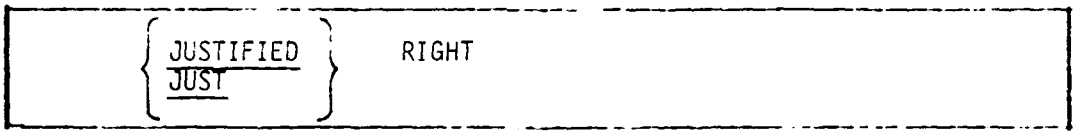
If VALUE is not specified, the initial value is unpredictable.

USACSC GUIDELINES. None.

2.4.7.21 JUSTIFIED CLAUSE.

FUNCTION. The JUSTIFIED clause specifies non-standard positioning of data within a receiving alphabetic or alphanumeric data item.

FORMAT.




SYNTAX RULES.

- The JUSTIFIED clause is applicable only at the elementary item level.
- JUST is an abbreviation for JUSTIFIED.
- This clause may only be specified for fixed-length alphabetic or alphanumeric data items.
- This clause may not be used with 88-level data items.

GENERAL RULES.

- The JUSTIFIED clause is only applicable when it is in a data description entry which is the receiving field of a MOVE instruction.
- Alignment of alphabetic and alphanumeric data in the receiving field of a MOVE occurs as follows:

This type of data is normally aligned within an elementary data item beginning in the leftmost position and moving character by character toward the right. Space filling or truncation occurs to the right.

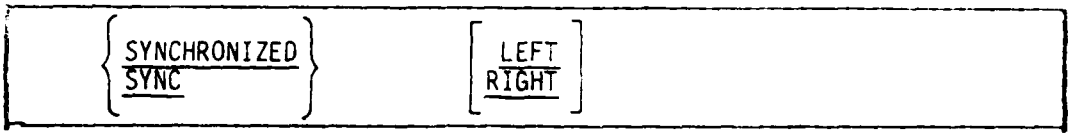
If JUSTIFIED is specified, the alphabetic or alphanumeric data is aligned beginning in the rightmost position and moving character by character toward the left. Space filling or truncation occurs to the left. If JUSTIFIED is specified and the receiving field is larger than the sending data item, the data is aligned at the rightmost character position in the data item with space fill for the leftmost character positions.

USACSC GUIDELINES. None.

2.4.7.22 SYNCHRONIZED CLAUSE.

FUNCTION. The SYNCHRONIZED clause specifies the alignment of an elementary item on one of the proper boundaries of core storage. It is used to insure efficiency when performing arithmetic operations on an item.

FORMAT.




SYNTAX RULES.

- SYNC is an abbreviation of SYNCHRONIZED and is the preferred form.

- This clause is designed to be used only with elementary items. Some vendors also allow SYNC to be used with Ø1-level record descriptions.

GENERAL RULES.

- This clause reserves the storage area between the leftmost and rightmost natural boundaries for this data item and aligns it within these boundaries. If the data item is smaller than the reserved space, slack bytes are inserted to fill the unused positions.

- If slack bytes are added, they do not affect the size of the elementary item. They are included in the size of the group item to which the synchronized elementary item belongs.

- If the LEFT or RIGHT options are not specified, the vendor determines alignment within the natural boundaries. The vendor may override these options.

  If LEFT is specified, the item will be aligned beginning from left to right.

  If RIGHT is specified, the item will be aligned beginning from right to left.

- If the data description contains an operational sign, the sign will appear in its normal operational position regardless of whether SYNC RIGHT or SYNC LEFT is specified.

- Whenever a SYNC item is referenced in the source program, the original PICTURE size of the item is used in determining any action which depends on size, such as truncation or justification.

2.4.7.22 SYNCHRONIZED CLAUSE. (Cont.)

- Where a SYNC clause is used within the scope of an OCCURS clause, each occurrence of the item is synchronized.

- When a SYNC clause is used with an item which contains a REDEFINES clause, the item that is being redefined must have the proper boundary alignment for the item that is redefining it.

- This clause is hardware dependent and each vendor specifies the application of the clause.

VENDORS' GUIDELINES.

- IBM.

IBM allows the use of SNYC with Ø1-level record descriptions as well as elementary items. If used at the Ø1-level, every elementary item within the record description is synchronized.

If either the LEFT or RIGHT option is specified, it is treated as comments.

The boundary alignment of a data item depends upon the USAGE clause specified.

1. For DISPLAY items, the SYNC clause is treated as comments.

2. For COMP items, the alignment and size of boundaries are determined by the size of the item.

a. If the PICTURE is in the range from S9 through S9(4), the item is aligned on a halfword (even) boundary.

b. If the PICTURE is in the range from S9(5) through S9(18), the item is aligned on a fullword (multiple of 4) boundary.

When SYNC is not specified for binary items, no space is reserved for slack bytes. Even if the COMP item is aligned, it is assumed to be unsynchronized and the computer generates coding to move the item to a properly aligned field before it is used in computation.

In the FILE SECTION, the compiler assumes that all level-Ø1 records containing synchronized items are aligned on a doubleword boundary in the buffer.

In the WORKING-STORAGE SECTION, the compiler aligns all level-Ø1 entries on a doubleword boundary.

In the LINKAGE SECTION, the compiler assumes that all level-Ø1 entries begin on a doubleword boundary. If the CALL statement with the USING option is used, all data items referenced by the calling and called programs must be correspondingly aligned.

2.4.7.22 SYNCHRONIZED CLAUSE. (Cont.)

USACSC GUIDELINES. Use of SYNC clause reduces generated coding and therefore run time for arithmetic operations and subscripting. To prevent addition of slack bytes all COMP SYNC items of the same type, such as counters, subscripts, totals etc., should be of the same length and grouped together under the same Ø entry.

2.4.7.23 BLANK WHEN ZERO CLAUSE.

FUNCTION. The BLANK WHEN ZERO clause is an editing feature which allows an item to be set to blanks when its value is zeroes.

FORMAT.

```
BLANK WHEN ZERO
```

SYNTAX RULES.

- This clause may only be used with an elementary numeric or numeric-edited item.
- This clause cannot be used for variable length items.

GENERAL RULES.

- If the value of the item is zero, it will contain only blanks.
- When this clause is used for an item which is numeric, the category of the item is considered to be numeric-edited.

USACSC GUIDELINES. None.

## 2.4.8 PROCEDURE DIVISION.

2.4.8.1 General Description. The PROCEDURE DIVISION must be included in every COBOL source program. This division specifies those procedures needed to solve a given problem. These procedures (computations, logical decisions, input/output, etc.) are expressed in meaningful statements, similar to English, which employ the concepts of verbs to denote actions, and statements and sentences to describe procedures.

2.4.8.2 Structure.

FORMAT.

```
PROCEDURE DIVISION [ USING identifier-1 [identifier-2] ...] .
 [DECLARATIVES.
 {section-name SECTION. USE Sentence
 [paragraph-name. [sentence] ...] ...} ...
END DECLARATIVES.]
 {section-name SECTION     [priority]
 [paragraph-name. [sentence] ...] ...} ...
```

TITLE. The PROCEDURE DIVISION title must begin with the words PROCEDURE DIVISION in Margin 'A' followed by a period. This must appear on a line by itself.

HEADER. The PROCEDURE DIVISION header is followed, optionally, by Declarative Sections, which are in turn followed by procedures, each made up of statements, sentences, paragraphs, and/or sections, in a syntactically valid format. The end of the PROCEDURE DIVISION (and the physical end of the program) is that physical position in a COBOL source program after which no further procedures appear.

STATEMENT. The statement is the basic unit of the PROCEDURE DIVISION. A statement is a syntactically valid combination of words and symbols beginning with a COBOL verb. There are three types of statements: conditional statements containing conditional expressions (that is, test for a given condition), imperative statements consisting of an imperative verb and its operands, and compiler-directing statements consisting of a compiler-directing verb and its operands.

2.4.8.2 Structure. (Cont.)

SENTENCE. A sentence is composed of one or more statements. The statements may optionally be separated by semicolons. A sentence must be terminated by a period followed by a space.

PARAGRAPH. Several sentences that convey one idea or procedure may be grouped to form a paragraph. A paragraph must begin with a paragraph-name followed by a period. A paragraph may be composed of one or more successive sentences. A paragraph ends immediately before the next paragraph-name or section-name, at the end of the PROCEDURE DIVISION, or, in the Declarative portion, at the key words END DECLARATIVES.

• PARAGRAPH AND SECTION-NAMES. Paragraphs should be kept short and modular, no more than one routine should be contained in the same paragraph, and paragraph-names should be meaningful and should be stated in terms of the application if possible.

• PARAGRAPH NUMBERING. A four digit number will be the first part of each paragraph or section-name followed by a hyphen and a title. Paragraphs will be in ascending order by this number. All paragraph numbers will be initially incremented by 10's or more to allow room for future insertions. If additional insertions are needed, use a hyphen and another number following the first four digit numbers to maintain sequence. Alphabetic characters forming a meaningful title will be included after the number as shown below:

```
0210-READ-INPUT-TRANSACTION.
```

SECTION. One or more paragraphs form a section. A section must begin with a section header (header-name followed by the word SECTION, followed by a period; if program segmentation is desired, a space and a priority number followed by a period may be inserted after the word SECTION). The general term procedure-name may refer to both paragraph-names and section-names.

USING PHRASE. The USING phrase is present if and only if the object program is to function under control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

• OPERANDS. Each of the operands of the USING phrase of the PROCEDURE DIVISION header must be defined as a data item in the LINKAGE SECTION of the program in which this header occurs. Also, it must have a 01 or 77-level number.

2.4.8.3 General Rules. The PROCEDURE DIVISION is organized with the following basic grouping of expressions:

2.4.8.3 General Rules. (Cont.)

1. Declaratives.
2. Statements.
3. Arithmetic Expressions.
4. Conditional Expressions.

2.4.8.3.1 Declaratives. DECLARATIVE SECTIONS must be grouped at the beginning of the PROCEDURE DIVISION preceded by the word DECLARATIVES. Declarative sections are concluded by the key words END DECLARATIVES. (For more information, see USE AFTER ERROR PROCEDURE.)

2.4.8.3.2 Statements. A STATEMENT is a syntactically valid combination of words and symbols beginning with a COBOL verb or the word IF followed by any appropriate operands, and other COBOL words that are necessary for the completion of the statement. There are three types of COBOL statements. They are: compiler-directing, imperative and conditional.

- CATEGORIES OF STATEMENTS. Refer to FIGURE 2-13.

| CATEGORY | VERBS |
|---|---|
| Arithmetic | ADD<br>COMPUTE<br>DIVIDE<br>INSPECT (TALLYING)<br>MULTIPLY<br>SUBTRACT |
| Compiler Directing | COPY<br>USE |
| Conditional | ADD (SIZE ERROR)<br>CALL<br>COMPUTE (SIZE ERROR)<br>DELETE (INVALID KEY)<br>DIVIDE (SIZE ERROR)<br>IF<br>MULTIPLY (SIZE ERROR)<br>READ (AT END or INVALID KEY)<br>RETURN (AT END)<br>REWRITE (INVALID KEY)<br>SEARCH (AT END)<br>START (INVALID KEY)<br>SUBTRACT (SIZE ERROR)<br>WRITE |

FIGURE 2-13

2.4.8.3.2 Statements. (Cont.)

| | |
|---|---|
| Data Movement | INSPECT (REPLACING)<br>MOVE |
| Ending | STOP |
| Input-Output | ACCEPT (identifier)<br>CLOSE<br>DELETE<br>DISPLAY<br>OPEN<br>READ<br>REWRITE<br>START<br>STOP (literal)<br>WRITE |
| Inter-Program Communicating | CALL<br>CANCEL |
| Ordering | RELEASE<br>RETURN<br>SORT |
| Procedure Branching | CALL<br>EXIT<br>GO TO<br>PERFORM |
| Table Handling | SEARCH<br>SET |

FIGURE 2-13 (Cont.)

IF is a verb in the COBOL sense; it is recognized that it is not a verb in English.

• COMPILER-DIRECTING STATEMENT. A compiler-directing statement directs the compiler to take action at compilation time. A compiler-directing statement contains one of the compiler-directing verbs (COPY, USE) and its operands.

• IMPERATIVE STATEMENT. An imperative statement indicates a specific unconditional action to be taken by the object program. An imperative statement is any statement that is neither a conditional statement, nor a compiler-directing statement. An imperative statement may consist of a sequence of imperative statements, each possibly separated from the next by a separator. The imperative verbs are listed on FIGURE 2-14.

2.4.8.3.2 Statements. (Cont.)

| | | | |
|---|---|---|---|
| ACCEPT | DISPLAY | OPEN | START (2) |
| ADD (1) | DIVIDE (1) | PERFORM | STOP |
| CALL(3) | EXIT | READ (4) | SUBTRACT (1) |
| CANCEL | GO | RELEASE | WRITE (5) |
| CLOSE | INSPECT | REWRITE (2) | |
| COMPUTE (1) | MOVE | SET | |
| DELETE (2) | MULTIPLY (1) | SORT | |

(1) Without the optional SIZE ERROR phrase.
(2) Without the optional INVALID KEY phrase.
(3) Without the optional ON OVERFLOW phrase.
(4) Without the optional AT END phrase or INVALID KEY phrase.
(5) Without the optional INVALID KEY phrase or END-OF-PAGE phrase.

FIGURE 2-14

Whenever 'imperative-statement' appears in the General Format of statements described in this Chapter, 'imperative-statement' refers to that sequence of consecutive imperative statements that must be ended by a period or an ELSE phrase associated with a previous IF statement or a WHEN phrase associated with a previous SEARCH statement.

- CONDITIONAL STATEMENT. A conditional statement is a statement containing a condition that is tested to determine which of the alternate paths of program flow is to be taken. A conditional statement is one of the following:

  1. An IF, SEARCH or RETURN statement.
  2. A READ statement that specifies the AT END or INVALID KEY phrase.
  3. A WRITE statement that specifies the INVALID KEY or END-OF-PAGE phrase.
  4. A START, REWRITE or DELETE statement that specifies the INVALID KEY phrase.
  5. An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) that specifies the SIZE ERROR phrase.
  6. A RECEIVE statement that specifies a NO DATA phrase.
  7. A STRING, UNSTRING or CALL statement that specifies the ON OVERFLOW phrase.

2.4.8.3.3 Arithmetic Expressions.

- DEFINITION. An ARITHMETIC EXPRESSION can be any of the following:

  1. An identifier described as a numeric elementary item.

2.4.8.3.3 Arithmetic Expressions. (Cont.)

2. A numeric literal.

3. Those identifiers and literals separated by arithmetic operators.

4. Two arithmetic expressions separated by an arithmetic operator.

5. An arithmetic expression inclosed in parentheses.

Any arithmetic expression may be preceded by an unary operator.

Those identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

• ARITHMETIC OPERATORS. There are five binary arithmetic operators and two unary arithmetic operators that may be used in arithmetic expressions. They are represented by specific characters that must be preceded by a space and followed by a space. Refer to FIGURE 2-15.

| Binary Arithmetic Operator | Meaning |
|---|---|
| + | ADDITION |
| - | SUBTRACTION |
| * | MULTIPLICATION |
| / | DIVISION |
| ** | EXPONENTIATION (not supported by Honeywell) |
| **Unary Arithmetic Operator** | **Meaning** |
| + | The effect of multiplication by the numeric literal +1 |
| - | The effect of multiplication by the numeric literal -1 |

FIGURE 2-15

• PERMISSIBLE ARITHMETIC SYMBOL PAIRS. A symbol pair in an arithmetic expression is the occurrence of two symbols that appear in sequence.

Permissible Symbol Pairs. Refer to FIGURE 2-16.

2.4.8.3.3 Arithmetic Expressions. (Cont.)

| First Symbol \ Second Symbol | Variable (identifier or literal) | * / ** + - | Unary + or Unary - | ( | ) |
|---|---|---|---|---|---|
| Variable (identifier or literal) | - | P | | - | P |
| * / ** + - | P | - | P | P | - |
| Unary + or Unary - | P | - | P | P | - |
| ( | P | - | P | P | - |
| ) | - | P | | - | P |

P indicates a permissible pairing
- indicates that the pairing is not permitted

FIGURE 2-16

An arithmetic expression may begin only with a left parenthesis, a unary +, a unary -, or a variable, and may end only with a right parenthesis or a variable.

There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression.

- RELATION CONDITION.

  COMPARISON OF OPERANDS OF EQUAL SIZE.

  Characters in corresponding character positions of the two operands are compared from the high-order end through the low-order end. The high-order end is the leftmost position; the low-order end is the rightmost character position.

  If all pairs of characters compare equally through the last pair, the operands are considered equal when the low-order end is reached.

2.4.8.3.3 Arithmetic Expressions. (Cont.)

If a pair of unequal characters is encountered, the two characters are compared to determine their relative position in the collating sequence. The operand that contains the character higher in the collating sequence is considered to be the greater operand.

COMPARISON OF OPERANDS OF UNEQUAL SIZE.

If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by a sufficient number of spaces to make the operands of equal size in the case of alphabetic or alphanumeric items. For numeric operands of unequal size, comparison proceeds after extending the shorter operand on the left with a sufficient number of zeroes to make the operands of equal size.

COMPARISONS INVOLVING INDEX-NAMES AND/OR INDEX DATA ITEMS.

The comparison of two index-names is equivalent to the comparison of their corresponding occurrence numbers.

In the comparison of an index data item with an index-name or with another index data item, the actual values are compared without conversion.

The comparison of an index-name with a numeric item is permitted if the numeric item is an integer. The numeric integer is treated as an occurrence number. All other comparisons involving an index-name or index data item are not allowed. Refer to FIGURE 2-17 for Permissible Comparisons.

2.4.8.3.4 Arithmetic Operators. The arithmetic statements are used for computations. Individual operators are specified by the ADD, SUBTRACT, MULTIPLY and DIVIDE statements. Although the arithmetic statements are individual operators, they do have several common features.

1. The data descriptions of the operands need not be the same. Any necessary conversion and decimal point alignment is supplied throughout the calculation.

2. The maximum size of each operand is eighteen (18) decimal digits.

Also there are several options that are common to the arithmetic statement and appear frequently. A discussion of these options follows.

In the discussion that follows, a 'resultant-identifier' is that identifier associated with a result of an arithmetic operation.

2.4.8.3.4 Arithmetic Operators. (Cont.)

• GIVING OPTION. If the GIVING option is specified, the value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. This identifier, since not itself involved in the computation, may be a numeric-edited item.

• ROUNDED OPTION. If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the resultant-identifier, truncation will occur, and it will be relative to the size provided for the resultant-identifier. However, if ROUNDED is specified, the absolute value of the resultant-identifier is increased by one (1) whenever the most significant digit of the excess is greater than or equal to five (5).

2.4.8.3.4 Arithmetic Operators. (Cont.)

| First Operand \ Second Operand | GR | AL | AN | ANE | NE | FC* NNL | ZR NL | ED | BI | ID | EF | IF | SR | SN | IN | IDI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Group (GR) | NN | NN | NN | NN | NN | NN | NN | NN | NN | NN | NN | NN | NN | NN | | |
| Alphabetic (AL) | NN | NN | NN | NN | NN | NN | NN | NN | | | NN | | NN | NN | | |
| Alphanumeric (AN) | NN | NN | NN | NN | NN | NN | NN | NN | | | NN | | NN | NN | | |
| Alphanumeric Edited (ANE) | NN | NN | NN | NN | NN | NN | NN | NN | | | NN | | NN | NN | | |
| Numeric Edited (NE) | NN | NN | NN | NN | NN | NN | NN | NN | | | NN | | SR | NN | | |
| Figurative Constant (FC) * & Non-numeric Literal (NNL) | NN | NN | NN | NN | NN | | | NN | | | NN | | NN | NN | | |
| Fig. Constant ZERO (ZR) & Numeric Literal (NL) | NN | NN | NN | NN | NN | | | NU | NU | NU | NU | NU | NU | NU | IO[1] | |
| External Decimal (ED) | NN | NN | NN | NN | NN | NN | NU | NU | NU | NU | NU | NU | NN | NU | IO[1] | |
| Binary (BI) | NN | | | | | | NU | NU | NU | NU | NU | NU | | NU | IO[1] | |
| Internal Decimal (ID) | NN | | | | | | NU | NU | NU | NU | NU | NU | | NU | IO[1] | |
| External Floating Point (EF) | NN | NN | NN | NN | NN | NN | NU | NU | NU | NU | NU | NU | NN | NU | | |
| Internal Floating Point (IF) | NN | | | | | | NU | NU | NU | NU | NU | NU | | NU | | |
| Sterling Report (SR) | NN | NN | NN | NN | NN | NN | NN | NN | | | NN | | NN | NN | | |
| Sterling Nonreport (SN) | NN | NN | NN | NN | NN | NN | NU | NU | NU | NU | NU | NU | NN | NU | | |
| Index Name (IN) | | | | | | | IO[1] | IO[1] | IO[1] | IO[1] | | | | | IO | IV |
| Index Data Item (IDI) | | | | | | | | | | | | | | | IV | IV |

*FC includes all Figurative Constants except ZERO.
[1]Valid only if the numeric item is an integer.

NN = comparison as described for numeric operands
NU = comparison as described for numeric operands
IO = comparison as described for two index-names
IV = comparison as described for index data items

Permissible Comparisons

FIGURE 2-17

2.4.8.3.4 Arithmetic Operators. (Cont.)

When the low-order integer positions in a resultant-identifier are represented by the character 'P' (the assumed decimal scaling position) in the PICTURE for that resultant-identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

• SIZE ERROR OPTION. If, after decimal point alignment, the absolute value of a result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. The algebraic value of the final result of the arithmetic operation must be accurate to the precision specified by the resultant-identifier. The SIZE ERROR applies only to the final results of an arithmetic operation and does not apply to the intermediate results, except in MULTIPLY and DIVIDE statements in which case SIZE ERROR applies to intermediate results as well. (See VENDORS' GUIDELINES.) If the ROUNDED option is specified, rounding takes place before checking for size error. If a size error condition occurs, the subsequent action is a function of whether or not the SIZE ERROR option was specified.

1. Division by zero always causes a size error condition.

2. If the SIZE ERROR option is not specified and a size error condition occurs, the value of the resultant-identifier affected may be unpredictable.

3. If the SIZE ERROR option is specified and a size error condition occurs then the value of the resultant-identifier affected by the size error is not altered. After completion of the execution of the arithmetic operation, the imperative statement in the SIZE ERROR option is executed.

VENDORS' GUIDELINES.

• IBM. IBM does not allow ANSI specifications. The SIZE ERROR option never applies to intermediate results.

• FORMATION AND EVALUATION RULES.

Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first, and within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used, or parenthesized expressions are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

1st - Unary Plus and Minus.

2nd - Exponentiation.

3rd - Multiplication and Division.

4th - Addition and Subtraction.

2.4.8.3.4 Arithmetic Operators. (Cont.)

An arithmetic expression may only begin with the symbols '(', '+', '-', or a variable and may only end with ')' or a variable. There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression such that each left parenthesis is to the left of its corresponding right parenthesis.

2.4.8.3.5 Conditional Expressions. CONDITIONAL EXPRESSIONS enable the object program to select between alternate paths of control depending upon the truth value of the condition. Conditional expressions are specified in the IF, PERFORM and SEARCH statements.

2.4.8.3.6 Simple Conditions. The SIMPLE CONDITIONS are:

1. Relation Condition.
2. Class Condition.
3. Switch-status Condition.
4. Condition-name Condition.
5. Sign Condition.

A simple condition has a truth value of 'true' or 'false'. The inclusion in parentheses of simple conditions does not change the simple condition truth value.

• RELATION CONDITIONS. A relation condition causes a comparison of two operands, either of which may be an identifier or a literal.

The format for a relation condition is as follows:

```
{identifier-1              }   {IS [NOT] GREATER THAN}   {identifier-2              }
{literal-1                 }   {IS [NOT] LESS THAN   }   {literal-2                 }
{arithmetic-expression-1   }   {IS [NOT] EQUAL TO    }   {arithmetic-expression-2   }
```

1. The first operand is called the subject of the condition, the second operand is called the object of the condition.
2. The subject and object may not both be literals.
3. The subject and object must have the same USAGE, except when two numeric operands are compared.
4. A relation-operator, (IS NOT GREATER THAN, IS NOT LESS THAN, IS NOT EQUAL TO) specifies the type of comparison to be made.
5. A space must precede and follow each reserved word comprising the relational operator.

2.4.8.3.6 Simple Conditions. (Cont.)

- ABBREVIATED COMBINED RELATION CONDITIONS.

When simple or negated simple relation conditions are combined with logical connectives in a consecutive sequence such that a succeeding relation condition contains a subject or subject and relational operator that is common with the preceding relation condition, and no parentheses are used within such a consecutive sequence, any relation condition except the first may be abbreviated by:

1. The omission of the subject of the relation condition, or

2. The omission of the subject and relational operator of the relation condition.

The format for an abbreviated combined relation condition is:

relation-condition { { AND / OR } [NOT] [relational-operator] object } ...

Within a sequence of relation conditions both of the above forms of abbreviation may be used. The effect of using such abbreviations is as if the last preceding stated subject were inserted in place of the omitted subject, and the last stated relational operator were inserted in place of the omitted relational operator. The result of such implied insertion must comply with the rules for combinations of conditions, logical operators, and parentheses, shown in FIGURE 2-18 below. This insertion of an omitted subject and/or relational operator terminates once a complete simple condition is encountered within a complex condition.

| Given the following element | Location in conditional expression | | In a left-to-right sequence of elements: | |
|---|---|---|---|---|
| | First | Last | Element, when not first, may be immediately preceded by only: | Element, when not last, may be immediately followed by only: |
| simple-condition | Yes | Yes | OR, NOT, AND, ( | OR, AND, ) |
| OR or AND | No | No | simple-condition, ) | simple-condition, NOT, ( |
| NOT | Yes | No | OR, AND, ( | simple-condition, ( |
| ( | Yes | No | OR, NOT, AND, ( | simple-condition, NOT, ( |
| ) | No | Yes | simple-condition, ) | OR, AND, ) |

FIGURE 2-18

2.4.8.3.6 Simple Conditions. (Cont.)

The interpretation applied to the use of the word 'NOT' in an abbreviated combined relation condition is as follows:

(1) If the word immediately following 'NOT' is 'GREATER', '>', 'LESS', '<', 'EQUAL', '=', then the 'NOT' participates as part of the relational operator; otherwise

(2) The 'NOT' is interpreted as a logical operator and, therefore, the implied insertion of subject or relational operator results in a negated relation condition.

Some examples of abbreviated combired and negated combined relation conditions and expanded equivalents follow.

| Abbreviated Combined Relation Condition | Expanded Equivalent |
|---|---|
| a > b AND NOT < c OR d | ((a > b) AND (a NOT < c)) OR (a NOT < d) |
| a NOT EQUAL b OR c | (a NOT EQUAL b) OR (a NOT EQUAL c) |
| NOT a = b OR c | (NOT (a = b)) OR (a = c) |
| NOT (a GREATER b OR < c) | NOT ((a GREATER b) OR (a < c)) |
| NOT (a NOT > b AND c AND NOT d) | NOT ((((a NOT > b) AND (a NOT > c)) AND (NOT (a NOT > d)))) |

- COMPARISON OF NUMERIC OPERANDS.

  1. For those operands whose class is numeric, a comparison is made with respect to the algebraic value of the operands.

  2. The length of the literal is not significant.

  3. Zero is considered a unique value regardless of the sign.

  4. Unsigned numeric operands are considered positive for purposes of comparison.

- COMPARISON OF NON-NUMERIC OPERANDS.

  1. For non-numeric operands, or one numeric and one non-numeric operand, a comparison is made with respect to a specified collating sequence of characters. For EBCDIC collating sequence, reference COLLATING SEQUENCE in the Glossary.

  2. If one of the operands is described as numeric, it is treated as though it were moved to an alphanumeric data item of the same size as the numeric data item, and contents of this alphanumeric data item were then compared to the non-numeric operand. (See MOVE statement.)

2.4.8.3.6 Simple Conditions. (Cont.)

3. The size of an operand is the total number of characters in the operand.

4. Numeric and non-numeric operands may only be compared when their USAGE is the same, implicity or explicity.

- COMPARISON OF OPERANDS OF EQUAL SIZE.

Comparison proceeds by comparing characters in corresponding character positions starting from the high-order end (leftmost position) and continuing until either a pair of unequal characters is encountered, or the low-order end (rightmost character position) is reached, whichever comes first. If all pairs of characters compare equally through the last pair, the operands are considered equal when the low-order end is reached.

The first encountered pair of unequal characters is compared to determine their relative position in the collating sequence. The operand that contains the character that is positioned higher in the collating sequence is considered to be the greater operand.

- OPERANDS OF UNEQUAL SIZE. If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by a sufficient number of spaces to make the operands of equal size.

- COMPARISONS INVOLVING INDEX-NAMES AND/OR INDEX DATA ITEMS. Relation tests may be made between:

1. Two index-names: The result is the same as if the corresponding occurrence numbers were compared.

2. An index data item and an index-name or another index data item: The actual values are compared without conversion.

3. An index-name and an integer numeric item: The occurrence number that corresponds to the value of the index-name is compared to the integer numeric item.

4. The results of the comparison of an index data item with any data item not specified above is undefined, and therefore not allowed.

- CLASS CONDITIONS. The class condition determines whether the operand is numeric; that is, consists entirely of the characters '0', '1', '2', ..., '9', with or without operational signs, or alphabetic; that is, consist entirely of the characters 'A', 'B', 'C', ..., 'Z', space.

2.4.8.3.6 Simple Conditions. (Cont.)

The general format for the class condition is:

```
identifier    IS [NOT]  { NUMERIC    }
                        { ALPHABETIC }
```

The operand being tested must be described implicitly or explicitly, USAGE DISPLAY. (See VENDORS' GUIDELINES.)

The identifier being tested is determined to be numeric only if the contents consist of any combination of the digits Ø through 9. If the PICTURE of the identifier being tested does not contain an operational sign, the identifier being tested is determined to be numeric only if the contents are numeric and an operational sign is not present. If the PICTURE does contain an operational sign, the identifier being tested is determined to be numeric only if the contents are numeric and a valid operational sign is present.

The NUMERIC test cannot be used with an identifier described as alphabetic. For example, refer to FIGURE 2-19.

| PICTURE | VALUE | REPRESENTATION | TESTS NUMERIC |
|---|---|---|---|
| | | IBM | IBM |
| S999 | +123 | F1 F2 C3 | Yes |
| S999 | -123 | F1 F2 D3 | Yes |
| S999 | 123 | F1 F2 F3 | Yes |
| 999 | +123 | F1 F2 C3 | No |
| 999 | -123 | F1 F2 D3 | No |
| 999 | 123 | F1 F2 F3 | Yes |

FIGURE 2-19

The identifier being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters 'A' through 'Z' and the space.

The ALPHABETIC test cannot be used with an identifier described as numeric.

2.4.8.3.6 Simple Conditions. (Cont.)

VENDORS' GUIDELINES.

- IBM.

  1. The operand being tested can be described as USAGE COMP.
  2. Valid operational signs are hexadecimal F, C, and D.
  3. For numeric data items described with the SEPARATE SIGN clause, valid operational signs are hexadecimal 4E and 60.

- CONDITION-NAME CONDITION (CONDITIONAL VARIABLE). In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with the condition-name.

The general format for the condition-name is:

```
88 condition-name
```

An example of the condition-name condition is:

```
Ø5     PAY-STATUS                PIC X.
    88 GS-12 VALUE '1'.
    88 GS-13 VALUE '2'.
    88 GS-14 VALUE '3'.
```

PAY-STATUS is the conditional variable; GS-12, GS-13 and GS-14 are condition-names. Only one of the conditions specified by condition-name can be present for individual records in the file. In order to determine the pay status of the individual whose record is being processed, IF GS-12 ... can be coded, and its true or false evaluation determines the subsequent path the object program takes.

If the condition-name is associated with a range or ranges of values, then the conditional variable is tested to determine whether or not its value falls in this range, including the end value. The result of the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

2.4.8.3.6 Simple Conditions. (Cont.)

A condition-name is used in conditions as an abbreviation for the relation condition. This can be done since the associated condition-name is equal to only one of the values (or ranges of values) assigned to that conditional variable hence, IF PAY-STATUS EQUALS 1... would have the same effect as using the condition-name test IF GS-12...

- SIGN CONDITION.

The sign condition determines whether or not the algebraic value of an arithmetic expression (i.e., an item defined as numeric) is less than, greater than, or equal to zero.

The general format for a sign condition is:

```
arithmetic-expression IS [NOT] { POSITIVE }
                               { NEGATIVE }
                               { ZERO     }
```

When used, NOT and the next key word specify one sign condition that defines the algebraic test to be executed for truth value, e.g., 'NOT ZERO' is a truth test for a non-zero (positive or negative) value.

An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero. An unsigned field is always positive or zero.

- SWITCH-STATUS CONDITION. A switch-status condition determines the 'on' or 'off' status of an implementor-defined switch. The implementor-name and the 'on' or 'off' value associated with the condition must be named in the SPECIAL-NAMES paragraph of the Environment Division.

The general format for the switch-status condition is as follows:

```
condition-name
```

The result of the test is true if the switch is set to the specified position corresponding to the condition-name.

2.4.8.3.7 Compound (Complex) Conditions. Two or more simple conditions can be combined to form a COMPOUND (COMPLEX) CONDITION. Each simple condition is separated by one of the logical operators ('AND' and 'OR') or negating these conditions with logical negation (NOT).

- The logical operators and their meanings are shown on FIGURE 2-20.

| Logical Operator | Meaning |
|---|---|
| AND | Logical conjunction: The truth value is 'true' if both of the conjoined conditions are true; 'false' if one or both of the conjoined conditions are false. |
| OR | Logical inclusive: The truth value is 'true' if one or both of the included conditions are true; 'false' if both included conditions are false. |
| NOT | Logical negation or reversal of truth value: The truth value is 'true' if the condition is false; 'false' if the condition is true. |

FIGURE 2-20

Logical operations must be preceded by a space and followed by a space.

The following table on FIGURE 2-21 shows the relationships between the logical operators and simple conditions 'A' and 'B'.

2.4.8.3.7 Compound (Complex) Conditions. (Cont.)

| SIMPLE CONDITION / LOGICAL OPERATION | | IF A IS True AND B IS True | IF A IS False AND B IS True | IF A IS True AND B IS False | IF A IS False AND B IS False |
|---|---|---|---|---|---|
| THEN | | | | | |
| A AND B | | True | False | False | False |
| A OR B | | True | True | True | False |
| NOT (A AND B) | W | False | True | True | True |
| NOT A AND B | I L | False | True | True | True |
| NOT (A OR B) | L B | False | False | False | True |
| NOT A OR B | E | True | True | False | True |
| A AND NOT B | | False | False | True | False |
| A OR NOT B | | True | False | True | True |

FIGURE 2-21

- EVALUATION HIERARCHY.

Logical evaluation begins with the least inclusive pair of parentheses and proceeds to the most inclusive.

2.4.8.3./ Compound (Complex) Conditions. (Cont.)

If the order of evaluation is not specified by parentheses, then the order is:

1. Arithmetic expressions.
2. Relational operators.
3. NOT condition.
4. AND and its surrounding conditions, starting at the left and proceeding to the right.
5. OR and its surrounding conditions, proceeding from left to right.

• COMPOUND CONDITION STRUCTURE. Parentheses will always be used to specify the order in which compound conditions are to be evaluated. When a single relation test (e.g., A = B below) can determine the truth or falsity of a compound condition, that relation should be written first.

For example, the statement

```
IF A EQUALS B OR C EQUALS D AND E EQUALS F ...
```

will be written in the following format:

```
IF (A EQUALS B) OR ((C EQUALS D) AND (E EQUALS F)) ...
```

This expression will be evaluated in the following manner. Refer to FIGURE 2-22.

2.4.8.3.7 Compound (Complex) Conditions. (Cont.)

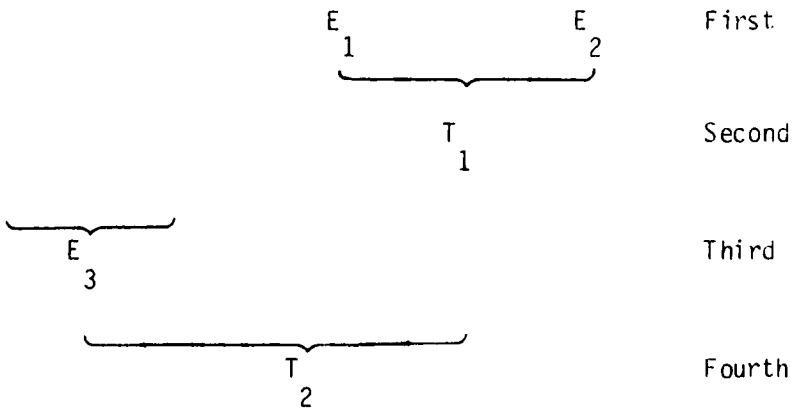
First - within the least inclusive parentheses (C EQUALS D) AND (E EQUALS F) the relational-operator EQUALS will be evaluated.

Second - The AND condition within that parenthetical expression will be evaluated.

Third - The relational-operator EQUALS of (A EQUALS B) will be evaluated.

Fourth - The OR function for the compound condition will be resolved.

IF (A EQUALS B) OR ((C EQUALS D) AND (E EQUALS F)) ...




where $E_n$ equals evaluation

$T_n$ equals total

FIGURE 2-22

## 2.4.9 STATEMENTS.

### 2.4.9.1 ACCEPT STATEMENT.

FUNCTION. The purpose/function of the ACCEPT statement is to obtain low volume data via a system input device.

FORMAT.

```
ACCEPT identifier [FROM mnemonic-name]
```

SYNTAX RULES.

- Identifier may be either a fixed-length group item or an elementary alphabetic, alphanumeric, or external decimal item. Identifier may not be any Special Register. The data is read and the appropriate number of characters is moved into the area reserved for identifier. No editing or error checking of the incoming data is done.

- If the input/output device specified by an ACCEPT statement is the same one designated for a READ statement, the results may be unpredictable.

GENERAL RULES.

- The ACCEPT statement causes the transfer of data from the hardware device. This data replaces the contents of the data item named by the identifier.

- The implementor will define, for each hardware device, the size of a data transfer.

- If a hardware device is capable of transferring data of the same size as the receiving data item, the transferred data is stored in the receiving data item.

- If a hardware device is not capable of transferring data of the same size as the receiving data item, then:

  a. If the size of the receiving data item exceeds the size of the transferred data, the transferred data is stored aligned to the left in the receiving data item. In Level 1, only one transfer of data is provided.

  b. If the size of the transferred data exceeds the size of the receiving data item, only the leftmost characters of the transferred data are stored in the receiving data item. The remaining characters of the transferred data which do not fit into the receiving data item are ignored.

2.4.9.1 ACCEPT STATEMENT. (Cont.)

- If the FROM phrase is not given, the device that the implementor specifies as standard is used.

VENDORS' GUIDELINES.

- IBM.

When an ACCEPT statement with the FROM CONSOLE option is executed, the following actions are taken:

1. A system generated message code is automatically displayed followed by the literal "AWAITING REPLY".

2. Execution is suspended. When a console input message is identified by the control program, execution of the ACCEPT statement is resumed and the message is moved to the specified identifier and left justified regardless of the PICTURE. If the field is not filled the low-order positions may contain invalid data.

USACSC GUIDELINES.

- The use of the FROM option with any other mnemonic-name than CONSOLE is not permitted. Use of the FROM CONSOLE is not permitted unless there is absolutely no other way to get the required information into the machine and that information is of such a nature that it can only be obtained from the operator in a realtime mode. Use of the FROM CONSOLE option must be justified on a case-by-case basis.

- The execution of the ACCEPT verb, in conjunction with the FROM option, is extremely inefficient, causes frequent error conditions due to erroneous replies and programers are therefore advised to utilize other methods in obtaining required data.

- Programers may utilize the ACCEPT verb, without the FROM option, to obtain data via the system logical input device.

- Another technique is to obtain the data via a READ statement after establishing the other related programing elements.

2.4.9.2 ADD STATEMENT.

FUNCTION. The ADD statement causes two or more numeric operands to be summed and the result to be stored.

FORMAT.

FORMAT 1.

```
ADD { identifier-1 } [ ,identifier-2 ] ... TO identifier-m [ ROUNDED ]
    { literal-1    } [ ,literal-2    ]

  [ identifier-n [ROUNDED] ] ... [ ;ON SIZE ERROR imperative-statement ]
```

FORMAT 2.

```
ADD { identifier-1 } , { identifier-2 } [ ,identifier-3 ] ...
    { literal-1    }   { literal-2    } [ ,literal-3    ]

    GIVING identifier-m [ ROUNDED ] [ , identifier-n [ROUNDED] ] ...

    [ ;ON SIZE ERROR imperative-statement ]
```

SYNTAX RULES.

- Each identifier must refer to an elementary numeric item except the object of the GIVING option (identifier-m) which may be a numeric edited data item.

- Each literal must be a numeric literal.

- The maximum size of each operand is 18 digits. The maximum size of the resultant sum, after decimal alignment, is 18 digits.

GENERAL RULES.

- In FORMAT 1, the values of the operands preceding the word TO are added together, then the sum is added to the current value of each identifier following the word TO (identifier-m, etc.). The result is stored in each resultant-identifier (identifier-m etc.).

2.4.9.2 ADD STATEMENT. (Cont.)

- In FORMAT 2, the values of the operands preceding the word GIVING are added together, then the sum is stored as the new value of the object of the GIVING option, identifier-m.

- The compiler insures that enough places are carried so as not to lose any significant digits during execution.

- The GIVING, ROUNDED and SIZE ERROR options are explained in Aritnmetic Options.

VENDORS' GUIDELINES.

- IBM.

  Only one identifier following the GIVING option in FORMAT 2 is allowed.

USACSC GUIDELINES.

- For efficient execution, and ease of maintenance, ADD statements should be as simple as possible. Consequently:

  Do not use ON SIZE ERROR unnecessarily. This option increases execution time and takes more space whether a size error exists or not.

  Do not use ROUNDED unnecessarily, for the same reason as above. ADD 5 and then MOVE to drop insignificant digits is more efficient.

  Avoid using more than 3 addends in a single ADD statement.

- In regard to the data items specified as operands in arithmetic statements:

  When they are all defined with the same USAGE, the relatively expensive operation of conversion is avoided. This does not apply to display items in the IBM-360/370. In these computers display items must be converted to packed decimal before they can be used arithmetically.

  When, for ADD and SUBTRACT, they are all defined with the same number of decimal places, the relatively expensive operation of scaling is avoided.

  When the data items are defined small enough so that the operation does not produce a calculated result greater than 15 digits the expensive operation of double precision arithmetic is avoided.

- In regard to a result item specified in the GIVING clause:

  When it is defined with sufficient integer places to provide for the maximum integer value possible, the need for the ON SIZE ERROR clause is eliminated.

2.4.9.2 ADD STATEMENT. (Cont.)

When it is defined with the same number of decimal places as the calculated result, the relatively expensive operation of truncation, rounding, or scaling (whichever applies in the particular case) is avoided.

When it is defined with the same USAGE as the calculated result, a conversion operation is avoided.

When it is defined as signed, the operation of removing the sign (which is generated automatically by the hardware) is avoided.

### 2.4.9.3 ALTER STATEMENT

FUNCTION. The ALTER statement is used to change the transfer point in a GO TO statement.

USACSC GUIDELINES. This statement is not to be used in maintenance or new development of programs. It is included here only for documentation purposes for previously coded modules. The complexity in program logic that the ALTER statement creates precludes its use due to maintenance consideration. As an alternative to using ALTER as a switching mechanism, a data item can be set and tested: Use a MOVE statement to change the value of a data item.

2.4.9.4 CALL STATEMENT.

FUNCTION. The CALL statement causes control to be transferred from one object program to another, within the run unit. This statement provides the communication link between a main object program and one or more subprograms.

FORMAT.

```
CALL literal-1 [ USING data-name-1 [ data-name-2 ] ... ]
```

SYNTAX RULES.

- Literal-1 must be a non-numeric literal.

- The USING option is included in the CALL statement (in the calling program) only if there is a USING phrase in the PROCEDURE DIVISION header of the called program. The order of the operands in each USING clause must be identical.

- Each of the operands in the USING clause must have been defined as a data item in the FILE SECTION, WORKING-STORAGE SECTION, or LINKAGE SECTION and must have a level-number of Ø1 or 77.

GENERAL RULES.

- The program whose name is specified by the value of literal-1 is the called program; the program in which the CALL statement appears is the calling program.

- The execution of a CALL statement causes control to pass from the calling program to the called program.

- A called program is in its initial state the first time it is called with a run unit. On all successive entries into the called program, the state of the program remains unchanged from its state when last existed.

- Unchanged will be data fields, file status, file positioning, and all alterable switches.

- It is the programer's responsibility to reinitialize such things as:

```
data items
PERFORM statements
ON statements
```

2.4.9.4 CALL STATEMENT. (Cont.)

- Called programs may contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program.

- The identifiers specified by the USING option of the CALL statement indicate those data items available to a calling program that may be referred to in the called program.

The order of appearance of the identifiers in the USING option of the CALL statement and the USING option in the PROCEDURE DIVISION header is critical. The data items in the USING options are paired on a one-to-one relationship.

Corresponding identifiers refer to a single set of data which is available and is defined in both the called and calling program. These data items correspond by position not by name. Their data descriptions must be equivalent.

In the case of index-names, no correspondence is established. Index-names in the called and calling program always refer to separate indices.

VENDORS' GUIDELINES.

- Honeywell.

  For the using list, data-name parameters are limited to 25.

USACSC GUIDELINES. Refer to USACSC COBOL programing techniques, "Transfer of Control" for further guidance.

2.4.9.5 CANCEL STATEMENT.

FUNCTION. To release memory areas occupied by the named program.

FORMAT.

```
CANCEL { literal-1    } [ ,literal-2    ] ...
       { identifier-1 } [ ,identifier-2 ]
```

SYNTAX RULES.

- Literal-1, literal-2, ...literal-n must each be a non-numeric literal whose value is a program-name.

- Identifier-1, identifier-2, ...identifier-n must each have a value that is a program-name.

GENERAL RULES.

- After the execution of a CANCEL statement, the CANCELed subprogram ceases to have a logical relationship to the run unit in which the CANCEL statement appears. When a subsequent CALL statement is executed naming the same program, that program begins execution in its initial state.

- A program cannot be cancelled which has been called but has not executed an EXIT PROGRAM statement.

- A logical relationship to a cancelled subprogram is reestablished only by executing a subsequent CALL.

- Control passes to the next statement when the program named in a CANCEL statement has not been called in this run unit or has already been CANCELed.

- If a CALLed program has not been CANCELed by a CANCEL statement, it is automatically CANCELed by the termination of the run unit of which it is a member.

- Memory areas associated with CANCELed subprograms are released for disposition by the operating system.

VENDORS' GUIDELINES. Not implemented in IBM DOS compiler and some levels of IBM OS compilers.

USACSC GUIDELINES. None.

### 2.4.9.6 CASE STATEMENT.

FUNCTION. The CASE statement is used to cause control to be passed to one or more imperative statements based on the value of an integer variable. Refer to "STRUCTURED PROGRAMMING STATEMENTS - MetaCOBOL Macro Facility" of the "Special Features" section.

2.4.9.7 CLOSE STATEMENT.

FUNCTION. The CLOSE statement terminates the processing of input-output reels, units, and files.

FORMAT.

```
CLOSE file-name-1 [{REEL}] [WITH {NO REWIND}]
                  [{UNIT}]       {LOCK     }
  [,file-name-2 [{REEL}] [WITH {NO REWIND}] ]]] ...
                 [{UNIT}]      {LOCK     }
```

SYNTAX RULES.

- The files referenced in the CLOSE statement need not all have the same organization or access.
- File-name must not be the name of a sort or merge file.
- The REEL or UNIT option may only be used for sequential files.

GENERAL RULES.

- A CLOSE statement may only be executed for a file in an open mode.
- The action taken if a file is in the open mode when a STOP RUN statement is executed is specified by the implementor.
- If a CLOSE statement has been executed for a file, no other statement can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

2.4.9.7 CLOSE STATEMENT. (Cont.)

• Following the successful execution of a CLOSE statement, the record area associated with file-name is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

Except where otherwise stated in the general rules below, the terms 'reel', 'unit', and 'volume' are synonymous and completely interchangeable in the CLOSE statement. Treatment of sequential mass storage files is logically equivalent to the treatment of a file on tape or analogous sequential media.

• For the purpose of showing the effect of various types of CLOSE statements as applied to various storage media, all files are divided into the following categories:

Unit record. A file whose input or output medium is such that rewinding, units and reels have no meaning.

Sequential single-volume. A sequential file entirely contained on one volume (one reel or one unit).

Sequential multi-reel/unit. A sequential file that is contained on more than one volume.

• The results of executing each type of CLOSE for each category of file are summarized below in FIGURE 2-23.

Relationship of Categories of Files and the Formats of the CLOSE Statement.

| CLOSE Statement Format | File Category | | | |
|---|---|---|---|---|
| | Non-Reel/Unit | Sequential Single-Reel/Unit | Sequential Multi-Reel/Unit | Non-Sequential Single/Multi-Reel/Unit |
| CLOSE | YES | YES | YES | YES |
| CLOSE WITH LOCK | YES | YES | YES | YES |

FIGURE 2-23

2.4.9.7 CLOSE STATEMENT. (Cont.)

| CLOSE Statement Format | File Category | | | |
|---|---|---|---|---|
| | Non-Reel/Unit | Sequential Single-Reel/Unit | Sequential Multi-Reel/Unit | Non-Sequential Single/Multi-Reel/Unit |
| CLOSE WITH NO REWIND | NO | NO | NO | NO |
| CLOSE REEL/UNIT | NO | NO | YES | NO |
| CLOSE REEL/UNIT FOR REMOVAL | NO | NO | YES | NO |
| CLOSE REEL/UNIT WITH NO REWIND | NO | NO | NO | NO |

FIGURE 2-23 (Cont.)

• Relationship of Categories of Files and the Formats of the CLOSE Statement. The definitions for FIGURE 2-23 are given below. Where the definition depends on whether the file is an input, output or input-output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

A. Previous Reels/Units Unaffected.

Input Files and Input-Output Files

All reels/units in the file prior to the current reel/unit are processed according to the implementor's standard reel/unit swap procedure, except those reels/units controlled by a prior CLOSE REEL/UNIT statement. If the current reel/unit is not the last file, the reels/units in the file following the current one are not processed.

Output Files

All reels/units in the file prior to the current reel/unit are processed according to the implementor's standard reel/unit swap procedure, except those reels/units controlled by a prior CLOSE REEL/UNIT statement.

2.4.9.7 CLOSE STATEMENT. (Cont.)

B. No Rewind of Current Reel.

The current reel/unit is left in its current position.

C. Close File.

Input Files and Input-Output Files (Sequential Access Mode):

If the file is positioned at its end and label records are specified for the file, the labels are processed according to the implementor's standard label convention. The behavior of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. If specified by the USE statement, a user's label procedure is executed. The order of execution of these two procedures is specified by the USE statement. In addition, other closing operations specified by the implementor are executed. If the file is positioned at its end and label records are not specified for the file, label processing does not take place but other closing operations specified by the implementor are executed. If the file is positioned at other than its end, the closing operations specified by the implementor are executed, but there is no ending label processing.

Input Files and Input-Output Files (Random or Dynamic Access Mode); Output Files (Random, Dynamic, or Sequential Access Mode):

If label records are specified for the file, the labels are processed according to the implementor's standard label convention. The behavior of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. If specified by the USE statement, a user's label procedure is executed. The order of execution of these two processes is specified by the USE statement. In addition, other closing operations specified by the implementor are executed. If label records are not specified for the file, label processing does not take place but other closing operations specified by the implementor are executed.

D. Reel/Unit Removal.

An implementor-defined technique is supplied to ensure that the current reel or unit is rewound when applicable and that the operating system is notified that the current reel or unit is logically removed from this run unit; however, the reel or unit may be accessed again, in its proper order of reels or units within the file, if a CLOSE statement without the REEL or UNIT phrase is subsequently executed for this file followed by the execution of an OPEN statement for the file.

2.4.9.7 CLOSE STATEMENT. (Cont.)

E. File Lock.

An implementor-defined technique is supplied to ensure that this file cannot be opened again during this execution of this run unit.

F. Close Reel/Unit.

Input Files:

The following operations take place:

1. A reel/unit swap.

2. The standard beginning reel/unit label procedure and the user's beginning reel/unit label procedure (if specified by the USE statement) are executed. The order of execution of these two label procedures is specified by the USE statement. The next executed READ statement for that file makes available the next data record on the new reel/unit.

Output Files and Input-Output Files:

The following operations take place:

1. (For output files only.) The standard ending reel/unit label procedure and the user's ending reel/unit label procedure (if specified by the USE statement) are executed. The order of execution of these two procedures is specified by the USE statement.

2. A reel/unit swap.

3. The standard beginning reel/unit label procedure and the user's beginning reel/unit label procedure (if specified by the USE statement) are executed. The order of execution of these two procedures is specified by the USE statement. For input-output files, the next executed READ statement that references that file makes the next logical data record on the next mass storage unit available. For output files, the next executed WRITE statement that references that file directs the next logical data record to the next reel/unit of the file.

G. Rewind.

The current reel or analogous device is positioned at its physical beginning.

H. Illegal.

This is an illegal combination of a CLOSE option and a file category. The results at object time are undefined.

2.4.9.7 CLOSE STATEMENT. (Cont.)

• The action taken if a file is in the open mode when a STOP RUN statement is executed is specified by the implementor. The action taken for a file that has been opened in a called program and not closed in that program prior to the execution of a CANCEL statement for that program is also specified by the implementor.

• If the OPTIONAL clause has been specified for the file in the FILE-CONTROL paragraph of the Environment Division and the file is not present, the standard end-of-file processing is not performed for that file.

• If a CLOSE statement without the REEL or UNIT phrase has been executed for a file, no other statement (except the SORT or MERGE statements with the USING or GIVING phrases) can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

• All reports associated with a report file that have been initiated must be ended with the execution of a TERMINATE statement before a CLOSE statement is executed for that report file.

• The WITH NO REWIND and FOR REMOVAL phrases will have no effect at object time if they do not apply to the storage media on which the file resides.

• Following the successful execution of a CLOSE statement, without the REEL or UNIT phrase, the record area associated with file-name is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

VENDORS' GUIDELINES.

• IBM.

• Files left open will be automatically closed by IBM OS, except under MVT, a file must be closed before STOP RUN or EXIT PROGRAM statement is executed. Failure to do this results in an abnormal termination.

• Use of I-O areas defined in FD's after CLOSE, before an OPEN or in the case of an INPUT file, after OPEN and before READ, are prohibited under DOS and under OS MVT. Conversion problems encountered when converting from DOS to OS precludes the use of I-O areas in the cases cited.

Each CLOSE statement for a file requires the use of a storage area that is directly proportional to the number of files being closed. Closing more than one file with the same statement is faster than when using a separate statement for each file. However, separate statements require less storage.

2.4.9.7 CLOSE STATEMENT. (Cont.)

USACSC GUIDELINES.

• CLOSE Statement. There are two ways in which to use the CLOSE statement when closing several files:

```
CLOSE DETAIL-FILE MASTER-FILE.

or

CLOSE DETAIL-FILE.

CLOSE MASTER-FILE.
```

• Tape files should be closed as soon as possible after end of file so that rewind action can begin immediately.

## 2.4.9.8 COMPUTE STATEMENT.

FUNCTION. The COMPUTE statement assigns to one or more data items the value of an arithmetic expression.

FORMAT.

```
COMPUTE identifier-1 [ROUNDED] [ , identifier-2 [ROUNDED] ] ...

    = arithmetic-expression [; ON SIZE ERROR imperative-statement]
```

SYNTAX RULES.

- Identifiers that appear only to the left of = must refer to either an elementary numeric item or an elementary numeric edited item.

- The arithmetic-expression option permits the user to combine arithmetic operations without the restrictions imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE.

- The assignment operator "=" (equal sign) must be used in the COMPUTE statement.

2.4.9.8 COMPUTE STATEMENT. (Cont.)

GENERAL RULES. Maximum size of each operand is 18 bytes.

VENDORS' GUIDELINES.

- Exponentiation may be accomplished to a fractional power.

| EXAMPLE: A ** .5 | or | A ** (B / D) |
|---|---|---|

This situation should be avoided as it requires floating point instructions. It would be more efficient to accomplish this operation with the DIVIDE and MULTIPLY statements.

USACSC GUIDELINES. The use of the COMPUTE statement generates more efficient coding than does the use of individual arithmetic statements because the compiler can keep track of internal work areas and does not have to store the results of intermediate calculations. It is the user's responsibility, however, to insure that the data is defined with the level of significance required in the answer. The compute statement should not be used for simple arithmetic operations.

- Hierarchy.

The compiler has a built-in hierarchy of operations. Example of *Hierarchy:*

| | | |
|---|---|---|
| ** | EXPONENTIATION | 1. |
| * | MULTIPLICATION | 2. |
| / | DIVISION | |
| + | ADDITION | 3. |
| - | SUBTRACTION | |

The sequence that takes place is to scan the expression from left to right three or more times.

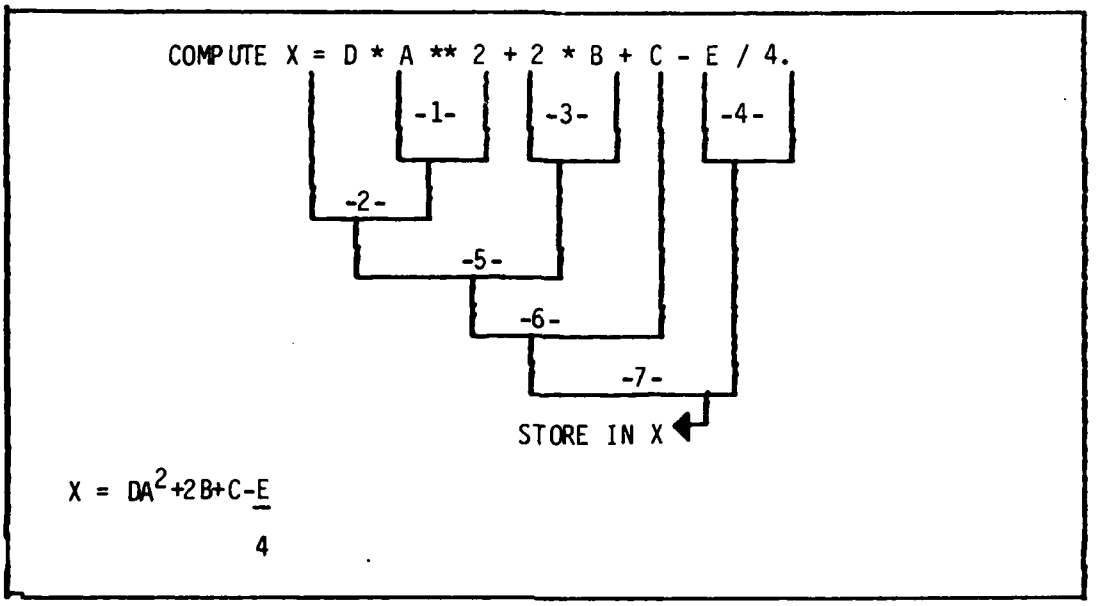
1. The first scan is to resolve any exponentiation.

2.4.9.8 COMPUTE STATEMENT. (Cont.)

2. The second scan across the expression from left to right will create instructions to take care of all multiplication and/or division, as they are encountered.

3. The third time through, instructions will be generated to accomplish the addition and/or subtraction.

- Example of a compute statement and what actually is accomplished.



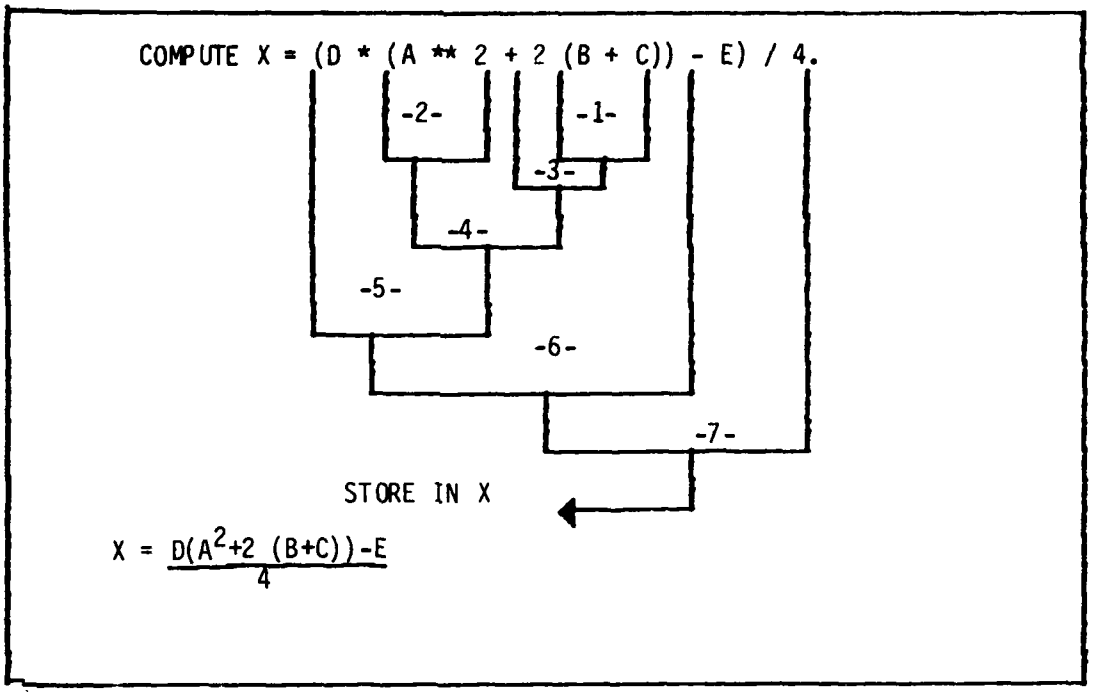

- We do have the capability of altering the COMPUTE statement to force a different evaluation to occur. We do this by use of parentheses.

If parentheses are used, the compiler will resolve the statement beginning with the innermost set of parentheses first and work its way out.

Within each set of parentheses, the scan is still left to right and uses the normal hierarchy.

- Let's take a look at this same COMPUTE statement with a few parentheses added.

2.4.9.8 COMPUTE STATEMENT. (Cont.)




- The COMPUTE statement may be used similar to FORTRAN notation.

```
EXAMPLE:    COMPUTE X = Y
            COMPUTE X = 0
```

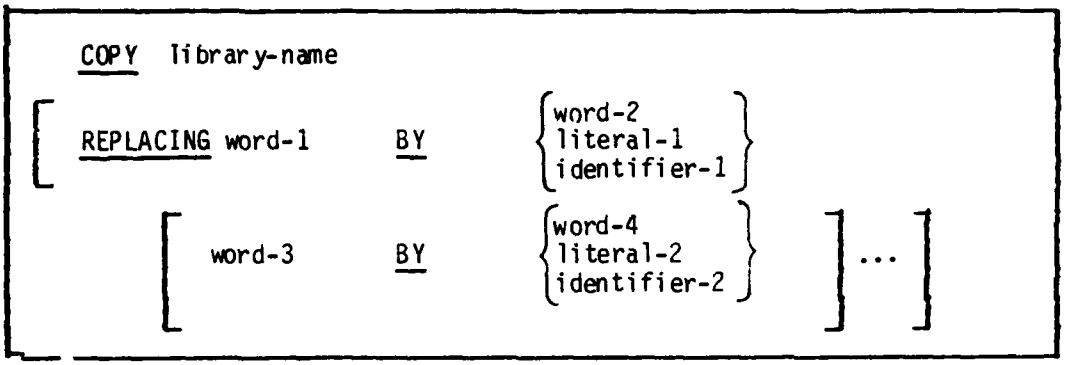
It is more efficient to use other statements such as MOVE for this purpose in COBOL.

- Arithmetic expressions may be used only with the COMPUTE statement and certain testing statements which we will get to shortly. The arithmetic operators may not be used with the other four arithmetic verbs.

## 2.4.9.9 COPY STATEMENT.

FUNCTION. The COPY statement allows the inclusion of prewritten entries in a source program. These may be DATA DIVISION entries, ENVIRONMENT DIVISION clauses, and/or PROCEDURE DIVISION procedures.

FORMAT.




SYNTAX RULES.

- The COPY statement must be preceded by a space and terminated by a period.
- Within a COBOL library, each library-name must be unique.
- A COPY statement may occur in the source program anywhere a character string or separator may occur. However, a COPY statement cannot be contained within another COPY statement.
- No other statement or clause may appear in the same entry as the COPY statement.
- Word-1, word-2, etc., may be a data-name, procedure-name, condition-name, mnemonic-name or file-name.
- The entries on the library itself remain unchanged even if the REPLACING option is used.
- If the REPLACING option is not specified, the library text is copied unchanged.
- If the REPLACING option is specified each properly matched identifier, word, or literal in the library text is replaced by the corresponding identifier, word, or literal.
- Text replacement occurs in the following manner:

Any separator (comma, space, semicolon) preceding the leftmost library text-word is copied into the source program.

2.4.9.9 COPY STATEMENT. (Cont.)

Starting with the leftmost library text-word, the entire REPLACING operand is compared to an equivalent number of contiguous lib text-words.

A match occurs if the library text is equal, characte for character, to the REPLACING operands.

If no match occurs, the comparison is continued using the next set of REPLACING operands until a match is found or there are no more REPLACING operands.

When all the sets of REPLACING operands have been compared and no match has occurred, the leftmost library text-word is copied into the source program. The next library text-word is then considered as the leftmost library text-word and the comparison cycle starts again.

When a match occurs between the REPLACING operands and the library text, the corresponding BY operands are placed into the source program. The comparison starts again with the library text-word immediately following the last text-word which participated in the match.

The comparisons continue until the library text-words have been depleted.

- A word or literal that is the result of the operation of REPLACING phrase cannot be operated on by any other REPLACING phrase in any explicit COPY statement.

GENERAL RULES.

- Compiling a source program containing a COPY statement is logically equivalent to processing all COPY statements prior to the processing of the resultant source program.

- The copied statements associated with text-name logically replace the entire COPY statement beginning with the resolved word COPY and ending with the punctuation character, "." period.

- Comment lines appearing in library text are copied into the source program unchanged.

- The text produced as a result of the complete processing of a COPY statement must not contain a COPY statement.

- Debugging lines are permitted within library text. If a COPY statement is specified on a debugging line, then the text that is the result of the processing of the COPY statement will appear as though it were specified on debugging lines with the following exception: comment lines in library text will appear as comment lines in the resultant source program.

- The syntactic correctness of the library text cannot be independently determined. The syntactic correctness of the entire COBOL source program cannot be determined until all COPY statements have been completely processed.

2.4.9.9 COPY STATEMENT. (Cont.)

- Library text must conform to the rules for COBOL reference format.

VENDORS' GUIDELINES.

- IBM.

  The SUPPRESS option is an IBM extension to the ANSI COBOL standards.

- Honeywell.

  The REPLACING option is not available.

USACSC GUIDELINES.

- The IBM SUPPRESS option, COPY library-name SUPPRESS, may be used to indicate that the library entry is not to be listed.

- The REPLACING option will not be used to alter Standard Data Elements and Codes.

- Qualification of names in the COBOL source program will not be allowed except when used in conjunction with a COPY statement.

## 2.4.9.10 USE FOR DEBUGGING STATEMENT.

FUNCTION. The USE FOR DEBUGGING statement identifies the items in the source program that are to be monitored by the associated debugging declarative procedure.

FORMAT.

```
section-name SECTION [priority-number].
USE FOR DEBUGGING ON procedure-name-1
```

SYNTAX RULES. All debugging sections must be written together immediately after the DECLARATIVES header. Except for the USE FOR DEBUGGING sentence itself, within the debugging procedure there must be no reference to any nondeclarative procedure.

GENERAL RULES.

- Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

- A debugging section is not executed for a specific operand more than once as the result of the execution of a single statement, no matter how many times the operand is explicitly specified. An exception to this rule is that each specification of a subscripted or indexed identifier will cause invocation of the debugging declarative. For a PERFORM statement that causes repeated execution of a procedure, any associated procedure-name debugging declarative section is executed once for each repetition.

- For debugging purposes, each separate occurrence of an imperative verb within an imperative statement begins a separate statement.

- Statements appearing outside the debugging sections must not refer to procedure-names defined within the debugging sections.

- Except for the USE FOR DEBUGGING sentence itself, statements within a debugging declarative section may refer to procedure procedure-names defined in a different USE procedure, only through the PERFORM statement.

- Procedure-names within debugging declarative sections must not appear in any USE FOR DEBUGGING sentence.

- Procedure-name-1 may appear in only one USE FOR DEBUGGING sentence, and only once in that sentence.

- When a USE FOR DEBUGGING operand is used as a qualifier, such a reference in the program does not activate the debugging procedures. The debugging procedures are activated immediately before the procedure-name-1 referenced in the

2.4.9.10 USE FOR DEBUGGING STATEMENT. (Cont.)

USE FOR DEBUGGING is called in the normal logic of the program. The WITH DEBUGGING MODE clause is stated in the SOURCE-COMPUTER paragraph.

- References to the DEBUG-ITEM special register may only be made from within a debugging declarative procedure.

USACSC GUIDELINES. All USE FOR DEBUGGING statements are to be used during testing and are not to be used in operational programs.

2.4.9.11 DEBUG-ITEM Special Register.

FUNCTION. The DEBUG-ITEM special register provides information for a debugging declarative procedure. It provides information about the conditions causing debugging section execution.

FORMAT.

```
01  DEBUG-ITEM.
    02  DEBUG-LINE       PICTURE IS X(6).
    02  FILLER           PICTURE IS X  VALUE SPACE.
    02  DEBUG-NAME       PICTURE IS X(30).
    02  FILLER           PICTURE IS X  VALUE SPACE.
    02  DEBUG-SUB-1      PICTURE IS S9999 SIGN IS LEADING SEPARATE
                         CHARACTER.
    02  FILLER           PICTURE IS X  VALUE SPACE.
    02  DEBUG-SUB-2      PICTURE IS S9999  SIGN IS LEADING SEPARATE
                         CHARACTER.
    02  FILLER           PICTURE IS X  VALUE SPACE.
    02  DEBUG-SUB-3      PICTURE IS S9999  SIGN IS LEADING SEPARATE
                         CHARACTER.
    02  FILLER           PICTURE IS X  VALUE SPACE.
    02  DEBUG-CONTENTS   PICTURE IS X(n).
```

GENERAL RULES.

- Before each debugging section is executed, DEBUG-ITEM is filled with spaces. The contents of the DEBUG-ITEM subfields are then updated according to the rules for the MOVE statement, with one exception: DEBUG-CONTENTS is updated as if the move were an alphanumeric to alphanumeric elementary move without conversion of data from one form of internal representation to another. After updating, each field contains:

DEBUG-LINE: The source-statement sequence-number or the compiler-generated card number, depending on the compiler option chosen.

DEBUG-NAME: The first 30 characters of the name causing debugging section execution. Any qualifiers are separated by the word "OF." (Subscripts or indexes are not entered in DEBUG-NAME.)

DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3: If the DEBUG-NAME is subscripted or indexed, the occurrence number of each level is entered in the respective DEBUG-SUB-n. If the item is not subscripted or indexed, these fields remain spaces.

DEBUG-CONTENTS: Data is moved into DEBUG-CONTENTS as shown in FIGURE 2-24.

2.4.9.11 DEBUG-ITEM Special Register. (Cont.)

| Item causing debug section execution | DEBUG-LINE contains number of COBOL statement referring to | DEBUG-NAME contains | DEBUG-CONTENTS contains |
|---|---|---|---|
| GO TO procedure-name-n | GO TO statement | procedure-name-n | spaces |
| procedure-name-n in SORT INPUT/ OUTPUT PROCEDURE | SORT statement | procedure-name-n | "SORT INPUT" "SORT OUTPUT" as applicable |
| PERFORM statement transfer of control | this PERFORM statement | procedure-name-n | "PERFORM LOOP" |
| procedure-name-n in a USE procedure | statement causing USE procedure execution | procedure-name-n | "USE PROCEDURE" |
| implicit transfer from previous sequential procedure | previous statement executed in previous sequential procedure (see note) | procedure-name-n | "FALL THROUGH" |
| 1st execution of 1st nondeclarative procedure | line number of first nondeclarative pro-cedure-name | first nondeclarative procedure-name | "START PROGRAM" |
| NOTE: If this paragraph is preceded by a section header and control is passed through the section header, the statement number refers to the section header. | | | |

FIGURE 2-24

2.4.9.12 DELETE STATEMENT.

FUNCTION. The DELETE statement logically removes a record from a mass storage file.

FORMAT.

```
DELETE file-name RECORD [; INVALID KEY imperative-statement]
```

SYNTAX RULES.

- The INVALID KEY phrase must not be specified for a DELETE statement which references a file which is in sequential access mode.

- The INVALID KEY phrase must be specified for a DELETE statement which references a file which is not in sequential access mode and for which an applicable USE procedure is not specified.

GENERAL RULES.

- The associated file must be open in the I-O mode at the time of the execution of this statement.

- For files in the sequential access mode, the last input-output statement executed for file-name prior to the execution of the DELETE statement must have been a successfully executed READ statement. The MSCS logically removes from the file the record that was accessed by that READ statement.

- For a file in random access mode, the MSCS logically removes from the file that record identified by the contents of the RELATIVE KEY or prime record key data item associated with file-name. If the file does not contain the record specified by the key, an INVALID KEY condition exists.

- After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

- The execution of a DELETE statement does not affect the contents of the record area associated with file-name.

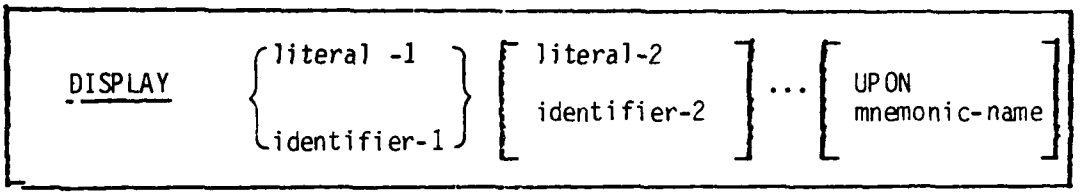
- The current record pointer is not affected by the execution of a DELETE statement.

- The execution of the DELETE statement causes the value of the specified FILE STATUS data item, if any, associated with file-name to be updated.

## 2.4.9.13 DISPLAY STATEMENT.

FUNCTION. The DISPLAY statement causes low volume data to be written to a specified output device.

FORMAT.




SYNTAX RULES.

- Each literal may be any figurative constant, except ALL literal.
- If a numeric literal is used, it must be an unsigned integer.

GENERAL RULES.

- If the UPON option is not used, the vendor specifies the default device which will be used.
- Identifier may not be any special register.
- The size of the output records acceptable to each hardware device is designated by each vendor.
- The DISPLAY statement causes the contents of each operand to be transferred to the hardware device in the order listed. The size of the sending item equals the sum of the sizes of the operands.
- The implementor will define, for each hardware device, the size of a data transfer.
- If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.
- If the hardware device is not capable of receiving data of the same size as the data item being transferred, then one of the following applies.

a. If the size of the data item being transferred exceeds the size of the data that the hardware device is capable of receiving in a single transfer, the data beginning with the leftmost character is stored aligned to the left in the receiving hardware device. Only one transfer of data is provided.

b. If the size of the data item that the hardware device is capable of receiving exceeds the size of the data being transferred, the transferred data is stored aligned to the left in the receiving hardware device.

2.4.9.13 DISPLAY STATEMENT. (Cont.)

- When a DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes associated with the operands, and the values of the operands are transferred in the sequence in which the operands are encountered.

- The implementor's standard display device is used.

VENDORS' GUIDELINES.

- IBM.

An IBM extension allows the use of the DISPLAY statement options UPON CONSOLE, UPON SYSPUNCH (OS)/SYSPCH (DOS), or UPON SYSOUT (OS)/SYSLST (DOS).

IBM defines the size of the output records for its hardware devices to be 1ØØ (OS)/72 (DOS) characters for CONSOLE, 12Ø characters for SYSOUT (OS)/SYSLST (DOS) and 8Ø characters (72 usable characters and 8 program-name characters) for SYSPUNCH (OS)/SYSPCH (DOS).

IBM assigns SYSOUT/SYSLST as the default device if the UPON option is not used.

Identifiers with a USAGE COMP are automatically converted to external decimal format. Signed values cause a low-order sign overpunch to be developed; for example, -34 displays as 3M; +34 displays as 3D.

The following statements cause the printer to space before printing: DISPLAY and WRITE AFTER ADVANCING. A simple WRITE statement or a WRITE BEFORE ADVANCING causes the printer to space after printing. Mixing these two categories of output statements in the same program may cause overprinting.

- Honeywell.

Identifiers with USAGE COMP, COMP-1, COMP-2, and COMP-5 are not converted to printable characters when the DISPLAY is executed. Consequently, the data to be displayed should be moved to a USAGE DISPLAY elementary numeric field before the DISPLAY.

USACSC GUIDELINES.

- Informative type data will be displayed on the system printer.

- Use of the DISPLAY statement for messages requiring console-operator response will be avoided except for contingencies where no other approach is feasible. Approval must be obtained from QAD prior to use of this statement.

- Informative messages which do not require console operator intervention may be displayed to the console. However, it is preferable that such messages be directed to SYSLST (DOS) or SYSOUT (OS).

2.4.9.14 DIVIDE STATEMENT.

FUNCTION. The DIVIDE statement is used to find the quotient and remainder resulting from the division of one numeric data item into another numeric data item.

FORMAT.

FORMAT 1.

```
DIVIDE { identifier-1 } INTO identifier-2 [ ROUNDED ]
       { literal-1    }
    [ ON SIZE ERROR imperative-statement ]
```

FORMAT 2.

```
DIVIDE { identifier-1 } { INTO } { identifier-2 } GIVING identifier-3
       { literal-1    } { BY   } { literal-2    }
    [ ROUNDED ]  [REMAINDER identifier-4]
    [ ON SIZE ERROR imperative-statements ]
```

SYNTAX RULES.

- Each identifier must refer to an elementary numeric item except the identifiers associated with the GIVING or REMAINDER clause which may be a numeric-edited item.

- Each literal must be a numeric literal.

- The maximum size of each operand is 18 decimal digits. The maximum size of the decimal-aligned results of the division (quotient and remainder) may not exceed 18 decimal digits each.

GENERAL RULES.

- In FORMAT 1, the value of identifier-1 (or literal-1) is divided into the value of identifier-2. The result (quotient) of the division is stored in the dividend (identifier-2).

2.4.9.14 DIVIDE STATEMENT. (Cont.)

• In FORMAT 2, the value of identifier-1 (or literal-1) is divided into identifier-2 (or literal-2) or when BY is used the value of identifier-1 (or literal-1) is divided by identifier-2 (or literal-2). The result is stored in the object of the GIVING option, identifier-3. A remainder, if specified, is stored in the object of the REMAINDER option, identifier-4.

• A remainder is defined as the result of subtracting the product of the quotient and the divisor from the dividend. The remainder may be numeric-edited.

• When the ROUNDED option is used, the quotient is rounded after the remainder has been determined.

• When ON SIZE ERROR is used with REMAINDER the following rules pertain:

If the SIZE ERROR occurs in the quotient no REMAINDER calculation is meaningful. Thus the contents of both identifier-3 and identifier-4 REMAIN unchanged.

If the SIZE ERROR occurs in the REMAINDER the contents of identifier-4 REMAIN unchanged. However, the user must do his own analysis to realize which has occurred.

• The GIVING, ROUNDED and SIZE ERROR options are explained in the Arithmetic Operations Section in Language Element: PROCEDURE DIVISION.

*VENDORS' GUIDELINES.*

• IBM.

IBM 360/370 requires display items to be converted into packed decimal before they can be used arithmetically.

USACSC GUIDELINES.

• For efficient execution, DIVIDE statements should be as simple as possible. Consequently:

Test for zero divisor before execution.

Do not use ON SIZE ERROR unnecessarily. This option increases execution time and takes more space whether a size error exists or not.

Do not use ROUNDED unnecessarily, for the same reason as above. To ADD 5 and then MOVE to drop insignificant digits is more efficient.

Use of GIVING option saves instructions.

• In regard to the data items specified as operands in arithmetic statements:

When they are all defined with the same USAGE, the relatively expensive operation of conversion is avoided, except on the IBM 360/370.

When, for ADD and SUBTRACT, they are all defined with the same number of decimal places, the relatively expensive operation of scaling is avoided.

2.4.9.14 DIVIDE STATEMENT. (Cont.)

- In regard to a result item specified in the GIVING clause:

When it is defined with sufficient integer places to provide for the maximum integer value possible, the need for the ON SIZE ERROR clause is eliminated.

When it is defined with the same number of decimal places as the calculated result, the relatively expensive operation of truncation, rounding, or scaling (whichever applies in the particular case) is avoided.

When it is defined with same USAGE as the calculated result, a conversion operation is avoided.

When it is defined as signed, the operation of removing the sign (which is generated automatically by the hardware) is avoided.

2.4.9.15 DO STATEMENT.

FUNCTION. The DO statement is used to cause the execution of the module specified by a procedure name. Refer to "STRUCTURED PROGRAMMING STATEMENTS - MetaCOBOL Macro Facility" of the "Special Features" section.

2.4.9.16 DO UNTIL STATEMENT.

FUNCTION. The DO UNTIL statement is used to cause execution of one or more imperative statements over and over again until a specified condition is met. Refer to "STRUCTURED PROGRAMMING STATEMENTS - MetaCOBOL Macro Facility" of the "Special Features" section.

2.4.9.17 DO WHILE STATEMENT.

FUNCTION. The DO WHILE statement is used to cause the execution of the statement repeatedly in a loop as long as the condition is true. Refer to "STRUCTURED PROGRAMMING STATEMENTS - MetaCOBOL Macro Facility" of the "Special Features" section.

2.4.9.18 ENTER STATEMENT.

FUNCTION. The ENTER statement provides a means of allowing the use of more than one language in the same program.

FORMAT.

```
ENTER language-name [ routine-name ]
```

SYNTAX RULES.

- The language-name is specified by the implementor and refers to any programing language which the implementor specifies may be entered through COBOL.

- A routine-name is a COBOL word and may be referred to only in an ENTER sentence.

- The sentence ENTER COBOL must follow the last non-COBOL statement to indicate to the compiler the resumption of COBOL source coding.

GENERAL RULES.

- The non-COBOL statements are executed in the object program as if they had been compiled into the object program following the ENTER statement.

- Details on non-COBOL languages and how they are to be written will be specified by individual vendors.

- If the statements in the entered language cannot be written in-line, a routine-name is given to identify the portion of the non-COBOL coding to be executed at this point in the procedure sequence. If the non-COBOL statements can be written in-line, routine-name is not used.

VENDORS' GUIDELINES. IBM uses this statement only as comments. The IBM compiler allows no other source language in-line in the source program. Non-COBOL routines can be incorporated by using the CALL statement.

USACSC GUIDELINES. This statement is not to be used in maintenance or new development of programs. It is included here only for documentation purposes for previously coded modules. The ENTER statement provides no useful function with the current IBM compiler and thus should be avoided.

2.4.9.19 EXIT STATEMENT.

FUNCTION. The EXIT statement provides the common end point for a series of procedures. The EXIT PROGRAM statement marks the logical end of a called program.

FORMAT.

```
paragraph-name.
     EXIT.
     EXIT PROGRAM.
```

SYNTAX RULES.

• The EXIT or EXIT PROGRAM statement must appear in a sentence by itself and must be the only sentence in the paragraph.

GENERAL RULES.

• The EXIT statement serves to allow the user to assign a procedure-name to a given point in a program. Usually control is passed through a series of procedures by going from paragraphs or sections sequentially or as directed. In the case of a PERFORM, control may be passed to another point in the program by associating a procedure-name at the point of the EXIT statement.

• If control reaches an EXIT paragraph and no associated PERFORM is active, control is passed through the EXIT statement to the first sentence in the next paragraph.

• The EXIT PROGRAM option is used in a subprogram (called by another program). If a program has been called and an EXIT PROGRAM statement is encountered, control is returned to the calling program at a point immediately following the CALL statement.

• If the program has not been called and an EXIT PROGRAM is encountered, it drops through to the first sentence in the next paragraph.

USACSC GUIDELINES.

• The EXIT PROGRAM statement should be used to return to the calling program in lieu of the GOBACK statement. The EXIT PROGRAM is recognized as a CODASYL standard whereas GOBACK is not.

• An EXIT paragraph should always be the last paragraph in a PERFORM (paragraph-1) through (paragraph-n) statement.

2.4.9.20 GO TO STATEMENT.

FUNCTION. The GO TO statement causes control to be transferred from one part of the PROCEDURE DIVISION to another.

FORMAT.

FORMAT 1.

```
GO TO procedure-name
```

FORMAT 2.

```
GO TO procedure-name-1    [ procedure-name-2 ]  ...
                            DEPENDING ON identifier
```

SYNTAX RULES.

- Identifier is the name of a numeric elementary item with no decimal positions.

- If FORMAT 1 of the GO TO statement appears in a series of imperative statements in a sentence, it must be the last statement in the sentence.

GENERAL RULES.

- In FORMAT 1, when the GO TO statement is executed, control is transferred to procedure-name-1.

- In FORMAT 2, control is transferred to one of a series of procedures depending on the value of identifier. If identifier is 1, control passes to procedure-name-1. If identifier is 2, control passes to procedure-name-2, etc.

Identifier must represent a positive unsigned integer. If not, the GO TO statement is ignored and control passes to the next statement in the normal sequence for execution.

VENDORS' GUIDELINES. IBM limits the number of procedure-names specified in FORMAT 2 to 2,031.

USACSC GUIDELINES. Use of the GO TO without procedure-name(s) or DEPENDING options is prohibited as the ALTER statement must be used for proper execution.

2.4.9.21 IF STATEMENT.

FUNCTION. The IF statement causes a condition to be evaluated. The subsequent action of the object program depends upon whether the condition is true or false.

FORMAT.

```
IF condition { statement-1   } { ELSE   statement-2   }
             { NEXT SENTENCE } { ELSE   NEXT SENTENCE }
```

SYNTAX RULES.

- Statement-1 and statement-2 represent an imperative statement.

- The phrase ELSE NEXT SENTENCE may be omitted if and only if it immediately precedes the period for the sentence.

GENERAL RULES. When an IF statement is executed, the following transfers of control occur.

- If the condition is true, statement-1 is executed if specified. If statement-1 contains a procedure branching statement, control is explicitly transferred in accordance with the rules for that statement. If there is no procedure branching statement, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

- If the condition is true and the NEXT SENTENCE is specified instead of statement-1, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

- If the condition is false, statement-1 or its surrogate NEXT SENTENCE is ignored, and statement-2, if specified, is executed. If statement-2 contains a procedure branching statement, control is explicitly transferred according to the rules for that statement. If statement-2 does not contain a procedure branching statement, control passes to the next executable sentence.

- If the condition is false and the ELSE NEXT SENTENCE phrase is specified, statement-1 is ignored, if specified, and control passes to the next executable sentence.

VENDORS' GUIDELINES.

- IBM.

  The ALL figurative constant should be used to test a single character field.

2.4.9.21 IF STATEMENT. (Cont.)

USACSC GUIDELINES.

- The nested IF statement will only be used on projects defined as Structured Programming by the Command. This does not apply to programs written prior to the published date of this change until they are rewritten. Nested IF's beyond three levels are discouraged.

- When statement-1 or statement-2 contains an IF statement, the IF statement is said to be nested.

- IF statements within IF statements may be considered as paired IF and ELSE combinations, proceeding from left to right. Any ELSE encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE.

- When control is transferred to the next sentence, implicitly or explicitly, control passes to the next sentence as written or to a return mechanism of a PERFORM or a USE statement.

- The following example contains two independent nests of conditional statements. The first nest ends after the statement PERFORM procedure-name-2; the second nest consist of the remainder of the sentence and has an implied ELSE NEXT SENTENCE before the period. 'A', 'B', 'C', 'D', 'E', and 'F' each corresponds to a conditional expression.

```
IF A THEN
  IF B THEN
     PERFORM PROCEDURE-NAME-1
  ELSE
     NEXT SENTENCE
ELSE
  IF C THEN
     NEXT SENTENCE
  ELSE
     PERFORM PROCEDURE-NAME-2
IF D THEN
  PERFORM PROCEDURE-NAME-3
  IF E THEN
     PERFORM PROCEDURE-NAME-4
     IF F THEN
          PERFORM PROCEDURE-NAME-5
     ELSE
          PERFORM PROCEDURE-NAME-6
ELSE
   STOP RUN.
```

2.4.9.21 IF STATEMENT. (Cont.)

IMPLIED NEXT SENTENCE.

• Compound conditions which require combining both the logical connectors AND and OR will not be permitted without the use of parentheses to make the exact evaluation of the total expression clear. Failure to use parentheses in this situation relies on the evaluation algorithm of the particular compiler(s) employed, which can vary from compiler to compiler, and is easily misunderstood and erroneously programed.

• Complex relational conditions which employ NOT logic must not, under any conditions, use the logical connector OR to connect the arguments of the relation condition. Such a construction will never work properly, as the imperative statement (the true path) will always be executed no matter what arguments are being tested. The proper logical connector is AND, and is the only one which will work properly. It is used in the following sense: "If condition to-be-tested is not equal to any of the following arguments, execute imperative statement." An example of improper coding is as follows:

```
IF A NOT EQUAL TO B OR C OR D PERFORM ...
```

The same statement, coded properly, is as follows:

```
IF A NOT EQUAL TO B AND C AND D PERFORM ...
```

The following example does not fall under the restriction stated in this paragraph, as it is simply a compound condition having two elements.

```
IF A NOT EQUAL TO B OR C NOT EQUAL TO D PERFORM ...
```

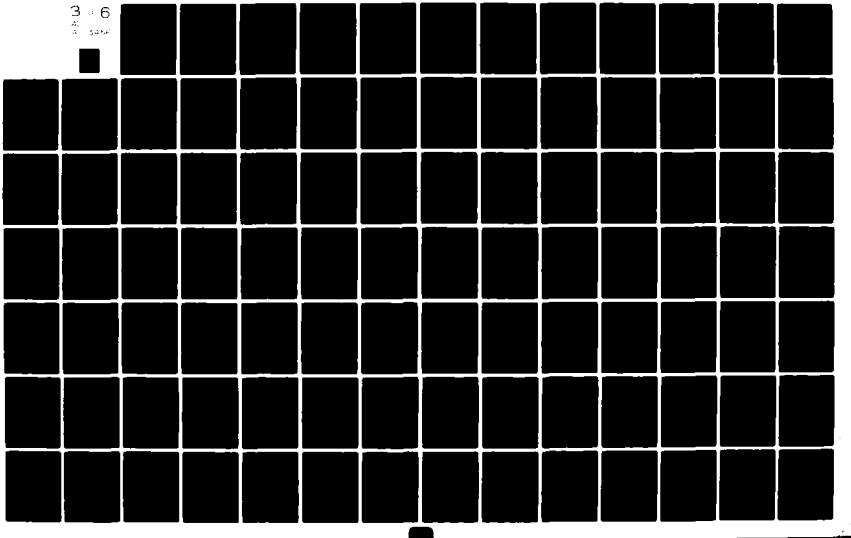
The above example is correct because the conditions being tested are different. However, if the second condition is changed to read "A NOT EQUAL TO D", it is wrong for the same reason the first example will always be executed.

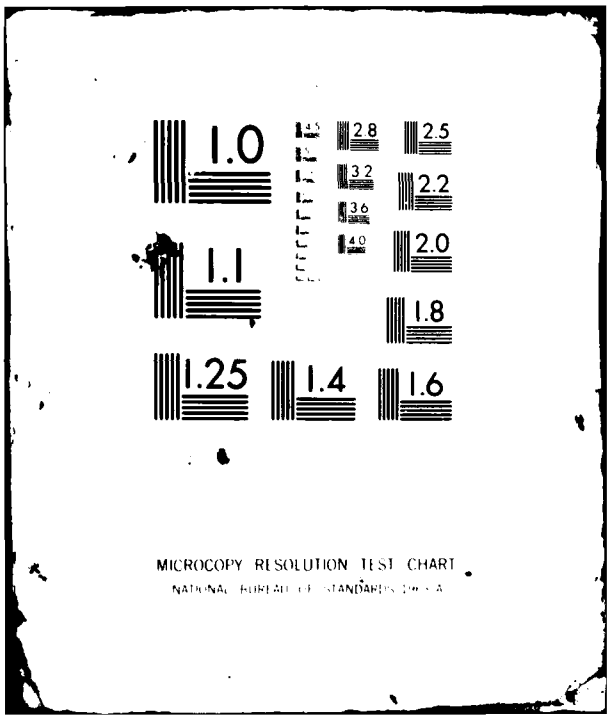
• Avoid:

Using the class test condition unnecessarily. This test is expensive in terms of coding generated and execution time.

AD-A113 456 ARMY COMPUTER SYSTEMS COMMAND FORT BELVOIR VA F/G 9/2
PROGRAMING PROCEDURES MANUAL (PPM).(U)
DEC 81

UNCLASSIFIED NL




1.0
1.1
1.25
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
1.4
1.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

2.4.9.21 IF STATEMENT. (Cont.)

Numeric comparisons of items with different numbers of decimal places. This type of comparison is particularly dangerous because comparison is based on composite field size; not minimum field size. For example, comparison of items defined as 9V99 and 9V9 cannot be equal unless the former has a zero as its low-order digit (e.g., 1.21 is not equal to 1.2). To circumvent this problem, decimal places can be truncated by moving the longer item to one properly defined (in this case as 9V9). Any conversion to a more efficient usage can be accomplished at the same time.

Comparing non-numeric items of different sizes.

Making group comparisons on records that contain slack positions, because the contents of the slack positions are generally unpredictable and can cause an invalid comparison.

Using the phrase "ELSE NEXT SENTENCE", which is meaningless unless it is within a nested IF statement. In a simple IF this is the result when the condition is not met; therefore, use of the phrase is redundant and should be avoided.

- The following techniques should be employed whenever possible to assure efficient IF statements. Refer to the table following this test for example references (FIGURE 2-25).

Do not specify a GO TO (in an IF statement) to a sequence of coding which returns control to the statement following the IF. If the sequence is 3 statements or less, place them in the IF statement (refer to example #1). If it is a longer sequence PERFORM it from the IF statement.

NEXT SENTENCE (or its equivalent) can often be conveniently avoided by reversing the relation (refer to example #2).

There is a definite breakdown point (in terms of efficiency of object code) between the use of a series of IF statements and the GO TO DEPENDING ON, where both are applicable. The breakeven point is 4 IF statements. (Refer to example #3). A still more efficient corollary should be used when one particular value of the series tested is expected to be present a majority of the time. In this case, test for the particular value with an IF statement and use the GO TO DEPENDING as the false branch of the IF.

Avoid testing the same item for more than one value once the condition has been satisfied (refer to example #4).

When appropriate, test for ranges of values rather than each individual value (refer to example #5).

Use of IF ... ALPHABETIC conditional is extremely inefficient. Avoid use or specify IF NOT NUMERIC.

2.4.9.21 IF STATEMENT. (Cont.)

- Examples. Refer to FIGURE 2-25.

| EXAMPLE | MORE EFFICIENT METHOD | LESS EFFICIENT METHOD |
|---|---|---|
| #1 | IF A EQUALS B<br>MOVE X TO Y<br><br>PERFORM Z.<br><br>D. MOVE ...<br>.<br>.<br>. | IF A EQUALS B GO TO C.<br>D. MOVE ...<br>.<br>.<br>.<br>C. MOVE X TO Y<br>PERFORM Z<br><br>GO TO D. |
| #2 | IF A NOT EQUAL TO B<br>MOVE C TO D | IF A EQUALS B NEXT SENTENCE<br>ELSE MOVE C TO D. |
| #3 | GO TO P1, P2, ..P12<br>DEPENDING ON A | IF A EQUALS 1 GO TO P1<br>IF A EQUALS 2 GO TO P2<br>.<br>.<br>.<br>IF A EQUALS 12 GO TO P12 |
| #4 | IF A EQUALS 1 MOVE X TO B<br>GO TO C.<br>IF A EQUALS 7 MOVE Y TO B<br>GO TO C.<br>IF A EQUALS 9 MOVE Z TO B<br>GO TO C.<br>C. | IF A EQUALS 1<br>MOVE X TO B.<br>IF A EQUALS 7<br>MOVE Y TO B.<br>IF A EQUALS 9<br>MOVE Z TO B. |
| #5 | IF A LESS THAN 1 GO TO C.<br><br>IF A LESS THAN 5 GO TO B.<br><br>C. ... | IF A EQUALS 1 GO TO B.<br>IF A EQUALS 2 GO TO B.<br>IF A EQUALS 3 GO TO B.<br>IF A EQUALS 4 GO TO B.<br>C. ... |

FIGURE 2-25

NOTE: When the IF statement is used in structured programming it must be terminated by the keyword ENDIF. Refer to "STRUCTURED PROGRAMMING STATEMENTS - MetaCOBOL Macro Facility" of the "Special Features" section.

2.4.9.22 INSPECT STATEMENT.

FUNCTION. The INSPECT statement provides the ability to tally (FORMAT 1), replace (FORMAT 2), or tally and replace (FORMAT 3) occurrences of single characters or groups of characters in a data item.

FORMAT.

FORMAT 1.

```
INSPECT identifier-1 TALLYING
{ , identifier-2 FOR { , { ALL      } { identifier-3 } [ { BEFORE } ] }
                     { , { LEADING  } { literal-1    }   { AFTER  } }
                     { , { CHARACTERS }                             }
                     {
                       INITIAL [ { identifier-4 } ] }
                               [ { literal-2    } ]
                     }
```

FORMAT 2.

```
INSPECT identifier-1 REPLACING
{ CHARACTERS BY { identifier-6 } [ { BEFORE } ]   INITIAL [ { identifier-7 } ]
                { literal-4    }   { AFTER  }              [ { literal-5    } ]
{ { , { ALL     } , { identifier-5 } BY { identifier-6 } [ { BEFORE } ]
      { LEADING }   { literal-3    }    { literal-4    }   { AFTER  }
      { FIRST   }
  INITIAL [ { identifier-7 } ] } } }
          [ { literal-5    } ]
```

2.4.9.22 INSPECT STATEMENT. (Cont.)

FORMAT 3.

```
INSPECT identifier-1 TALLYING

{, identifier-2 {FOR , {{ALL}      {identifier-3}} [{BEFORE}
                       {{LEADING}  {literal-1   }}  {AFTER }
                       {CHARACTERS}

  INITIAL      {identifier-4}]]}}
               {literal-2   }

REPLACING

{CHARACTERS BY    {identifier-6}   [{BEFORE}      INITIAL  {identifier-7}]
                  {literal-4   }   {AFTER }               {literal-5   }
{{, {ALL}     {, {identifier-5}   BY  {identifier-6}   [{BEFORE}
   {{LEADING}    {literal-3   }       {literal-4   }    {AFTER }
   {FIRST   }

  INITIAL  {identifier-7}]}}}}
           {literal-5   }
```

SYNTAX RULES.

ALL FORMATS

- Identifier-1 must reference either a group item or any category of elementary items, described (either implicitly or explicitly) as usage is DISPLAY.

2.4.9.22 INSPECT STATEMENT. (Cont.)

• Identifier-3 ... identifier-n must reference either an elementary alphabetic, alphanumeric or numeric item described (either implicitly or explicitly) as usage is DISPLAY.

• Each literal must be non-numeric and may be any figurative constant, except ALL.

• In Level 1, literal-1, literal-2, literal-3, literal-4, and literal-5, and the data items referenced by identifier-3, identifier-4, identifier-5, identifier-6 and identifier-7 must be one character in length. The length restriction does not apply to Level 2.

FORMATS 1 and 3 ONLY

• Identifier-2 must reference an elementary numeric data item.

• If either literal-1 or literal-2 is a figurative constant, the figurative constant refers to an implicit one character data item.

FORMATS 2 AND 3 ONLY

• The size of the data referenced by literal-4 or identifier-6 must be equal to the size of the data referenced by literal-3 or identifier-5. When a figurative constant is used as literal-4, the size of the figurative constant is equal to the size of literal-3 or the size of the data item referenced by identifier-5.

• When the CHARACTERS phrase is used, literal-4, literal-5, or the size of the data item referenced by identifier-6, identifier-7 must be one character in length.

• When a figurative constant is used as literal-3, the data referenced by literal-4 or identifier-6 must be one character in length.

GENERAL RULES.

• (1) Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying and/or replacing) begins at the leftmost character position of the data item referenced by identifier-1, regardless of its class, and proceeds from left to right to the rightmost character position as described in general rules (4) through (6).

• (2) For use in the INSPECT statement, the contents of the data item referenced by identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 will be treated as follows:

2.4.9.22 INSPECT STATEMENT. (Cont.)

(a) If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 are described as alphanumeric, the INSPECT statement treats the contents of each such identifier as a character-string.

(b) If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 are described as alphanumeric edited, numeric edited or unsigned numeric, the data item is inspected as though it had been redefined as alphanumeric (see general rule (2)(a)) and the INSPECT statement had been written to reference the redefined data item.

(c) If any of the identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 are described as signed numeric, the data item is inspected as though it had been moved to an unsigned numeric data item of the same length and then the rules in general rule (2)(b) had been applied.

• (3) In general rules (4) through (11) all references to literal-1, literal-2, literal-3, literal-4, and literal-5 apply equally to the contents of the data item referenced by identifier-3, identifier-4, identifier-5, identifier-6, and identifier-7, respectively.

• (4) During inspection of the contents of the data item referenced by identifier-1, each properly matched occurrence of literal-1 is tallied (FORMATS 1 and 3) and/or each properly matched occurrence of literal-3 is replaced by literal-4 (FORMATS 2 and 3).

• (5) The comparison operation to determine the occurrences of literal-1 to be tallied and/or occurrences of literal-3 to be replaced, occurs as follows:

(a) The operands of the TALLYING and REPLACING phrases are considered in the order they are specified in the INSPECT statement from left to right. The first literal-1, literal-3 is compared to an equal number of contiguous characters, starting with the leftmost character position in the data item referenced by identifier-1. Literal-1, literal-3 and that portion of the contents of the data item referenced by identifier-1 match if, and only if, they are equal, character for character.

(b) If no match occurs in the comparison of the first literal-1, literal-3, the comparison is repeated with each successive literal-1, literal-3, if any, until either a match is found or there is no next successive literal-1, literal-3. When there is no next successive literal-1, literal-3, the character position in the data item referenced by identifier-1 immediately to the right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins again with the first literal-1, literal-3.

2.4.9.22 INSPECT STATEMENT. (Cont.)

(c) Whenever a match occurs, tallying and/or replacing takes place as described in general rules (8) and (10). The character position in the data item referenced by identifier-1 immediately to the right of the rightmost character position that participated in the match is now considered to be the leftmost character position of the data item referenced by identifier-1, and the comparison cycle starts again with the first literal-1, literal-3.

(d) The comparison operation continues until the rightmost character position of the data item referenced by identifier-1 has participated in a match or has been considered as the leftmost character position. When this occurs, inspection is terminated.

(e) If the CHARACTERS phrase is specified, an implied one character operand participates in the cycle described in paragraphs (5)(a) through (5)(d) above, except that no comparison to the contents of the data item referenced by identifier-1 takes place. This implied character is considered always to match the leftmost character of the contents of the data item referenced by identifier-1 participating in the current comparison cycle.

• (6) The comparison operation defined in general rule (5) is affected by the BEFORE and AFTER phrases as follows:

(a) If the BEFORE and AFTER phrase is not specified, literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates in the comparison operation as described in general rule (5).

(b) If the BEFORE phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1 from its leftmost character position up to, but not including, the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule (5) is begun. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.

(c) If the AFTER phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase may participate only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1 from the character position immediately to the right of the rightmost character position of the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1 and the rightmost character position of the data item referenced by

2.4.9.22 INSPECT STATEMENT. (Cont.)

identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule (5) is begun. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

• (7) The contents of the data item referenced by identifier-2 is not initialized by the execution of the INSPECT statement.

• (8) The rules for tallying are as follows:

(a) If the ALL phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one (1) for each occurrence of literal-1 matched within the contents of the data item referenced by identifier-1.

(b) If the LEADING phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one (1) for each contiguous occurrence of literal-1 matched within the contents of the data item referenced by identifier-1, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.

(c) If the CHARACTERS phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one (1) for each character matched, in the sense of general rule (5)(e), within the contents of the data item referenced by identifier-1.

FORMAT 2

• (9) The required words ALL, LEADING and FIRST are adjectives.

• (10) The rules for replacement are as follows:

(a) When the CHARACTERS phrase is specified, each character matched, in the sense of general rule (5)e, in the contents of the data item referenced by identifier-1 is replaced by literal-4.

(b) When the adjective ALL is specified, each occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4.

2.4.9.22 INSPECT STATEMENT. (Cont.)

(c) When the adjective LEADING is specified, each contiguous occurrence of literal-3 matched in the data item referenced by identifier-1 is replaced by literal-4, provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which literal-3 was eligible to participate.

(d) When the adjective FIRST is specified, the leftmost occurrence of literal-3 matched within the contents of the data item referenced by identifier-1, is replaced by literal-4.

FORMAT 3

• (11) A FORMAT 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written with one statement being a FORMAT 1 statement with TALLYING phrases identical to those specified in the FORMAT 3 statement, and the other statement being a FORMAT 2 statement with REPLACING phrases identical to those specified in the FORMAT 3 statement. The general rules given for matching and counting apply to the FORMAT 1 statement and the general rules given for matching and replacing apply to the FORMAT 2 statement.

EXAMPLES.

Following are six examples of the INSPECT statement:

```
INSPECT word TALLYING count FOR LEADING "L" BEFORE INITIAL "A", count-1 FOR
LEADING "A" BEFORE INITIAL "L".

    Where word = LARGE, count = 1, count-1 = 0.
    Where word = ANALYST, count = 0, count-1 = 1.

INSPECT word TALLYING count FOR ALL "L", REPLACING LEADING "A" BY "E" AFTER
INITIAL "L".

    Where word = CALLAR, count = 2, word = CALLAR.
    Where word = SALAMI, count = 1, word = SALEMI.
    Where word = LATTER, count = 1, word = LETTER.

INSPECT word REPLACING ALL "A" BY "G" BEFORE INITIAL "X".

    Where word = ARXAX, word = GRXAX.
    Where word = HANDAX, word = HGNDGX.

INSPECT word TALLYING count FOR CHARACTERS AFTER INITIAL "J" REPLACING ALL
"A" BY "B".

    Where word = ADJECTIVE, count = 6, word = BDJECTIVE.
    Where word = JACK, count = 3, word = JBCK.
    Where word = JUJMAB, count = 5, word = JUJMBB.
```

2.4.9.22 INSPECT STATEMENT (Cont.)

INSPECT word REPLACING ALL "X" BY "Y", "B" BY "Z", "W" BY "Q" AFTER INITIAL "R".

Where word = RXXBQWY, word = RYYZQQY.
Where word = YZACDWBR, word = YZACDWZR.
Where word = RAWRXEB, word = RAQRYEZ.

INSPECT word REPLACING CHARACTERS BY "B" BEFORE INITIAL "A".

word before: 1 2 X Z A B C D
word before: B B B B B A B C D

2.4.9.23 MOVE STATEMENT.

FUNCTION. The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas.

FORMAT.

FORMAT 1.

```
MOVE { identifier-1 } TO identifier-2 [,identifier-3] ...
     { literal      }
```

FORMAT 2.

```
MOVE { CORRESPONDING } identifier-1 TO identifier-2
     { CORR          }
```

SYNTAX RULES.

- Identifier-1 and literal represent the sending area; identifier-2, identifier-3, ..., represent the receiving areas.

- An index data item cannot appear as an operand of a MOVE statement.

GENERAL RULES.

- The data designated by literal or identifier-1 is moved first to identifier-2, then to identifier-3 (if specified), etc.

- The rules referring to identifier-2 also apply to other receiving areas. Any subscripting or indexing associated with identifier-2, ..., is evaluated immediately before the data is moved to the respective data item. Any subscripting or indexing associated with identifier-1 is evaluated only once, immediately before data is moved to the first of the receiving operands.

- Any move in which the sending and receiving items are both elementary items is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric-edited, alphanumeric-edited. Literals and figurative constants are categorized as follows:

  Numeric literals belong to the category numeric.

  Non-numeric literals belong to the category alphanumeric.

  The figurative constant ZERO (ZEROS, ZEROES) belong to the category numeric.

2.4.9.23 MOVE STATEMENT. (Cont.)

The figurative constant SPACE (SPACES) belongs to the category alphabetic.

All other figurative constants belong to the category alphanumeric.

- Data items from each group are considered corresponding when they have the same name and qualification, up to but not including identifier-1 and identifier-2.

- The following rules apply to an elementary move between categories:

A numeric-edited, alphanumeric-edited, or alphabetic data item or the figurative constant SPACE cannot be moved to a numeric or numeric-edited data item.

A numeric literal, a numeric data item, a numeric-edited data item, or the figurative constant ZERO must not be moved to an alphabetic data item.

A non-integer (containing an explicit decimal point) numeric literal or a non-integer numeric data item must not be moved to an alphanumeric or alphanumeric-edited data item.

- Any necessary conversion of data from one form of internal representation to another takes place during the move, along with any specified editing in the receiving item.

When an alphanumeric or alphanumeric-edited item is the receiving item, the following data conversion principles apply.

1. Alignment takes place from left to right as defined under the Standard Alignment Rules (Language Concepts), unless the JUSTIFIED clause specifies otherwise.

2. Unused character positions are filled with spaces.

3. If the size of the sending item is greater than the size of the receiving item, the receiving item will be filled from the left and excess characters will be truncated on the right, unless otherwise specified by the JUSTIFIED clause.

4. If the sending item is described as a signed numeric, the operational sign will not be moved.

When a numeric or numeric-edited item is the receiving item, the following data conversion principles apply.

1. Alignment by decimal point takes place as defined by Standard Alignment Rules (Language Concepts).

2. Unused positions are zero-filled unless otherwise specified by editing requirements.

2.4.9.23 MOVE STATEMENT. (Cont.)

3. When the receiving item is a signed numeric, the sign of the sending item is placed in the receiving item. If the sending item is unsigned, a positive sign is generated for the receiving item.

4. When the receiving item is an unsigned numeric, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.

5. If the sending item has more digits to the left or to the right of the decimal point than the receiving item can contain, the excess digits are truncated.

6. If the sending item is alphanumeric, it is moved as if it were an unsigned numeric integer.

7. If the sending item contains any non-numeric characters, the results are unpredictable.

When an alphabetic item is the receiving item, the following data conversion considerations apply.

1. Alignment takes place from left to right according to the Standard Alignment Rules (Language Concepts), unless otherwise specified by the JUSTIFIED clause.

2. If the sending item is greater than the receiving item, truncation will occur to the right unless otherwise specified by the JUSTIFIED clause.

- Any move that is not an elementary move is treated as if it were an alphanumeric to alphanumeric elementary move, except that no data conversion takes place. The receiving area is filled without consideration of individual elementary or group items contained within either the sending or receiving area.

- The following chart summarizes various types of MOVE statements. Refer to FIGURE 2-26.

2.4.9.23 MOVE STATEMENT (Cont.)

| Category of Sending Data Item | | Category of Receiving Data Item: Alphanumeric | Alphanumeric Edited Alphanumeric | Numeric Integer Numeric Non-Integer Numeric Edited |
|---|---|---|---|---|
| Alphabetic | | Yes | Yes | No |
| Alphanumeric | | Yes | Yes | Yes |
| Alphanumeric Edited | | Yes | Yes | Yes |
| Numeric | Integer | No | Yes | Yes |
| | Non-Integer | No | No | Yes |
| Numeric-Edited | | No | Yes | No |

FIGURE 2-26

USACSC GUIDELINES.

- For character moves:

Employ caution when using a group item as a receiving field because data is moved to a group item on a character basis without regard to individual items within the group.

When the receiving field is longer than the sending field the MOVE is less efficient than when the receiving field is shorter or the fields are of equal size.

Remember that when the receiving field is shorter, the excess characters in the sending field are ignored.

- Edited moves are extremely costly.

- Each reference to an unsigned numeric item as a receiving field requires an extra object instruction.

- With IBM 360 only; MOVE, READ ... INTO, WRITE ... FROM, or REWRITE ... FROM where receiving field exceeds 256 characters requires extra object instructions, avoid where possible.

2.4.9.23 MOVE STATEMENT. (Cont.)

• Performing a MOVE operation for an item greater than 256 bytes in length requires the generation of more instructions than are required for that of a MOVE operation for an item of 256 bytes or less.

• When a MOVE statement with the CORRESPONDING option is executed, data items are considered CORRESPONDING only if their respective data names are the same, including all implied qualification, up to, but not including the data-names used in the MOVE statement itself. For example,

```
    Ø1  AA                          Ø1  XX
        Ø3   BB                         Ø3   BB
             Ø5   CC                         Ø5   CC
             Ø5   DD                         Ø5   DD
        Ø3   EE                         Ø3   YY
             Ø5   FF                         Ø5   FF
```

The statement MOVE CORRESPONDING AA TO XX will result in moving CC and DD but not FF because FF of EE does not correspond to FF of YY.

The compiler assumes that the data being moved conforms to PICTURE and USAGE specifications. If it does not, dissimilar results will occasionally occur because of the different code generated for various sending and receiving fields.

NOTE: The other rules for MOVE CORRESPONDING, of course, must still be satisfied.

• Numerous times a programer will read a record, MOVE it to WORKING-STORAGE and check to see if he needs that record. It could be possible that a very small percentage of these records are processed causing unnecessary CPU time to MOVE these records to WORKING-STORAGE. MOVE statements with larger amounts of data are costly in terms of CPU time. The programer should check to see if he needs a record prior to moving it to WORKING-STORAGE.

2.4.9.24 MULTIPLY STATEMENT.

FUNCTION. The MULTIPLY statement is used to multiply one numeric data item by another numeric data item.

FORMAT.

FORMAT 1.

```
MULTIPLY { identifier-1 } BY identifier-2 [ ROUNDED ]
         { literal-1    }
      [;ON SIZE ERROR imperative-statement ]
```

FORMAT 2.

```
MULTIPLY { identifier-1 } BY { identifier-2 } GIVING identifier-3
         { literal-1    }    { literal-2    }
      [ ROUNDED ]
      [;ON SIZE ERROR imperative statement ]
```

SYNTAX RULES.

- Each identifier must refer to an elementary numeric item except the object of the GIVING option (identifier-3) which may be a numeric-edited item.
- Each literal must be a numeric literal.
- The maximum size of each operand is 18 decimal digits. The maximum size of each receiving field, after decimal alignment, is 18 decimal digits.

GENERAL RULES.

- In FORMAT 1, the value of identifier-1 (or literal-1) is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by the result of the multiplication.
- In FORMAT 2, the value of identifier-1 (or literal-1) is multiplied by identifier-2 (or literal-2) and the result is stored in the object of the GIVING option, identifier-3.
- The GIVING, ROUNDED and SIZE ERROR options are explained in Arithmetic Operations, PROCEDURE DIVISION.

2.4.9.24 MULTIPLY STATEMENT. (Cont.)

VENDORS' GUIDELINES.

- IBM.

Only one identifier is allowed as the object of the GIVING option.

For the IBM 360/370, when they are defined small enough so that the operation does not produce a calculated result greater than 15 digits the expensive operation of double precision arithmetic is avoided.

For the IBM 360/370, when it is defined as signed, the operation of removing the sign (which is generated automatically by the hardware) is avoided.

USACSC GUIDELINES.

- For efficient execution, MULTIPLY statements should be as simple as possible. Consequently:

Do not use ON SIZE ERROR unnecessarily. This option increases execution time and takes more space whether a size error exists or not.

Do not use ROUNDED if possible, for the same reason as above. To ADD 5 and then MOVE to drop insignificant digits is more efficient.

Use of GIVING option saves instructions.

- In regard to the data items specified as operands in arithmetic statements:

When they are all defined with the same USAGE, the relatively expensive operation of conversion is avoided. This does not apply to display items in the IBM 360/370. In these computers, display items must be converted to packed decimal before they can be used arithmetically.

When, for ADD and SUBTRACT, they are all defined with the same number of decimal places, the relatively expensive operation of scaling is avoided.

- In regard to a result item specified in the GIVING clause:

When it is defined with sufficient integer places to provide for the maximum integer value possible, the need for the ON SIZE ERROR clause is eliminated.

When it is defined with the same number of decimal places as the calculated result, the relatively expensive operation of truncation, rounding, or scaling (whichever applies in the particular case) is avoided.

When it is defined with the same USAGE as the calculated result, a conversion operation is avoided.

2.4.9.25 OPEN STATEMENT.

FUNCTION. The OPEN statement initiates the processing of input, output, and input-output files. It performs checking and/or writing of labels and other input-output operations.

FORMAT.

FORMAT 1 - Indexed I-O and Relative I-O.

```
OPEN { INPUT file-name-1    [, file-name-2] ... }
     { OUTPUT file-name-3   [, file-name-4] ... }  ...
     { I-O file-name-5      [, file-name-6] ... }
```

FORMAT 2 - Sequential I-O.

```
                 [REVERSED      ]                 [REVERSED      ]
OPEN { INPUT file-name-1 [WITH NO REWIND ] [, file-name-2 [WITH NO REWIND ] ] ... }
     {                                                                          }...
     { OUTPUT file-name-3 [WITH NO REWIND ] [, file-name-4 [WITH NO REWIND ] ] ... }
     { I-O file-name-5 [, file-name-6 ] ...                                     }
```

SYNTAX RULES.

- The file-name must be defined by a file description entry in the Data Division. All files referenced in the OPEN statement need not all have the same organization or access.

- The REVERSED and the NO REWIND phrases can only be used with sequential files.

- At least one of the options INPUT, OUTPUT, or I-O must be specified. However, there must be no more than one instance of each option in the same statement, although more than one file-name may be used with each option. These options may appear in any order.

GENERAL RULES.

- The successful execution of an OPEN statement determines the availability of the file, results in the file being in an open mode, and makes the associated record area available to the program.

- The I-O option pertains only to mass storage files.

2.4.9.25 OPEN STATEMENT. (Cont.)

• The OPEN statement must not specify a sort-file, but an OPEN statement must be specified for all other files. The OPEN statement for a file must be executed prior to the execution of any statement referencing the file, explicitly or implicity. A file can be opened with the INPUT, OUTPUT, and I-O phrases in the same program. However, a second OPEN statement for a file cannot be executed prior to the execution of a CLOSE statement for that file. The CLOSE statement cannot contain the REEL or UNIT phrases. The OPEN statement does not obtain or release the first data record. A READ or WRITE statement must be executed to obtain or release, respectively, the first data record.

• The OPEN statement causes the user's beginning label subroutine to be executed if one is specified by a USE sentence in the Declaratives Section.

• When the REVERSED option is specified, subsequent READ statements for the file make the data records of the file available in reversed order; that is, starting with the last record.

• When the REVERSED option is specified, execution of the OPEN statement causes the file to be positioned at the end of the file.

• For the table below, Permissible Statements, an "X" at an intersection indicates that the specified statement, used in the access mode given for that row, may be used with the relative file organization and the open mode given at the top of the column.

| File Access Mode | Statement | Open Mode | | |
|---|---|---|---|---|
| | | Input | Output | Input-Output |
| Sequential | READ | X | | X |
| | WRITE | | X | |
| | REWRITE | | | X |
| | START | X | | X |
| | DELETE | | | X |
| Random | READ | X | | X |
| | WRITE | | X | X |
| | REWRITE | | | X |
| | START | | | |
| | DELETE | | | X |

Permissible Statements

• If label records are specified for the file, the beginning labels are processed as follows:

2.4.9.25 OPEN STATEMENT. (Cont.)

a. When the INPUT phrase is specified, the execution of the OPEN statement causes the labels to be checked in accordance with the implementor's specified conventions for input label checking.

b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the labels to be written in accordance with the implementor's specified conventions for output label writing.

The behavior of the OPEN statement when label records are specified but not present, or when label records are not specified but are present, is undefined.

- For files being opened with the INPUT or I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. If no records exist in the file, the current record pointer is set such that the next executed FORMAT 1 READ statement for the file will result in an AT END condition.

- The I-O phrase permits the opening of a mass storage file for both input and output operations. Since this option implies the existence of the file, it cannot be used if the mass storage file is being initially created.

- When the I-O phrase is used, and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

The label is checked in accordance with the implementor's specified conventions for input-output label checking.

The user's label subroutine, if one is specified by a USE sentence, is executed or new labels are written in accordance with the implementor's specified conventions for input-output label writing.

The label is written.

Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At that time the associated file contains no data records.

VENDOR'S GUIDELINES.

- IBM.

The IBM compiler allows the REVERSED option to be used with a sequential multiple reel file.

2.4.9.25 OPEN STATEMENT. (Cont.)

When opening a file, under the Operating System (OS), the NO REWIND option may have no effect on file positioning. Any file positioning at OPEN time is controlled by the operating system, and the NO REWIND option may be overridden by JCL.

The DOS NO REWIND option will keep the file static and negate any positioning commands from the system.

The REVERSED and the NO REWIND options can be used only with a sequential single reel file or guideline. The REVERSED option cannot be used for a file containing mode 'V' records.

If the option is specified for a file containing mode 'U' records, doubleword boundary alignment of the logical record is obtained only if the length of the logical record is divisible by eight. If there is no doubleword boundary alignment for a record containing SYNCHRONIZED items, the record cannot be properly processed.

Use of I-O areas defined in FD's after CLOSE, before an OPEN or in the case of an INPUT file, after OPEN and before READ are prohibited under DOS and under OS MVT. Conversion problems encountered when converting from DOS to OS precludes the use of I-O areas in the cases cited.

USACSC GUIDELINES.

- It is best to specify as many files in the same OPEN statement as is logically possible. Although some additional storage is required, it is offset by a relatively significant savings in execution time.

- Execution of an OPEN statement for an input file does not make the record-area in the File Section available for subsequent operations. Statements that refer to the record area must not be executed until a READ has been executed for the file.

- Execution of an OPEN for an output file makes a record area available. Therefore, statements that refer to the record(s) of the file may be executed after the OPEN and before the first WRITE.

### 2.4.9.26 PERFORM STATEMENT.

FUNCTION. The PERFORM statement is used to explicitly transfer control to one or more procedures and to return control implicitly whenever execution of the specified procedure is complete.

FORMAT.

FORMAT 1.

```
PERFORM procedure-name-1 [ THRU procedure-name-2 ]
```

FORMAT 2.

```
PERFORM procedure-name-1 [ THRU procedure-name-2 ]
    { identifier-1 }
    { integer-1    } TIMES
```

FORMAT 3.

```
PERFORM procedure-name-1 [ THRU procedure-name-2 ]
    UNTIL condition-1
```

FORMAT 4.

```
PERFORM procedure-name-1 [THRU procedure-name-2]
VARYING { index-name-1 }      FROM { index-name-2 }
        { identifier-2 }           { literal-1    }
                                   { identifier-3 }
BY      { literal-2    }      UNTIL condition-1
        { identifier-4 }
[AFTER  { index-name-3 }      FROM { index-name-4 }
        { identifier-5 }           { literal-3    }
                                   { identifier-6 }
```

2.4.9.26 PERFORM STATEMENT. (Cont.)

FORMAT 4. (Cont.)

```
BY { literal-4    }                    UNTIL    condition-2
   { identifier-7 }

[ AFTER { index-name-5 }               FROM { index-6    }
        { identifier-8 }                    { literal-5  }
                                            { identifier-9 }
  BY { literal-6    }                  UNTIL    condition-3 ] ]
     { identifier-10 }
```

SYNTAX RULES.

- Each identifier represents a numeric elementary item described in the DATA DIVISION. In FORMAT 2, identifier-1 must be described as a numeric integer.
- Each literal represents a numeric literal.
- If an index-name is specified in the VARYING or AFTER phrase, the following applies:

  The identifier in the associated FROM and BY phrases must be an integer data item.

  The literal in the associated FROM phrase must be a positive integer.

  The literal in the associated BY phrase must be a non-zero integer.
- If an index-name is specified in the FROM phrase, the following applies:

  The identifier in the associated VARYING or AFTER phrase must be an integer data item.

  The identifier in the associated BY phrase must be an integer data item.

  The literal in the associated BY phrase must be an integer.
- The literal in the BY phrase must not be zero.
- Condition-1, condition-2, condition-3 may be any conditional expression.

GENERAL RULES.

- An identifier in the BY option must not have a zero value.

2.4.9.26 PERFORM STATEMENT. (Cont.)

• If an index-name is specified in the VARYING or AFTER option, and an identifier is specified in the associated BY option, the identifier must have a positive value.

• The THRU option will be used in all cases except when procedure-1 refers to the section-name. In this case, the THRU option may or may not be used.

• When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. An implicit transfer of control to the next executable statement following the PERFORM statement is established as follows:

If procedure-name-1 is a section-name and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.

If procedure-name-2 is specified and is a paragraph-name, then the return is after the last statement of the paragraph.

If procedure-name-2 is specified and it is a section-name, then return is after the last statement of the last paragraph in that section.

• There is no relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at procedure-name-1 going through procedure-name-2.

GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2.

If there is more than one logical path to the return point, procedure-name-2 must be the name of a paragraph consisting of an EXIT statement, to which all of these paths must lead.

• If control is passed to any procedures within the range of the PERFORM by means other than the execution of the PERFORM statement, the PERFORM statements are ignored and control passes normally through the last statement of the procedures.

• FORMAT 1 is the basic PERFORM statement. The procedure referred to by the PERFORM is executed once and then control passes to the next executable statement following the PERFORM statement.

• In FORMAT 2, the procedures are performed the number of times specified by integer-1 or identifier-1.

If, at the time of execution of the PERFORM statement, the value of identifier-1 or integer-1 is zero or negative, control passes to the statement following the PERFORM statement.

Once the PERFORM statement has been initiated, any references to identifier-1 cannot alter the number of times the procedures are to be executed.

2.4.9.26 PERFORM STATEMENT. (Cont.)

• In FORMAT 3, the specified procedures are performed until the condition specified by the UNTIL option is used.

When the condition is true, control is passed to the statement following the PERFORM statement.

If the condition is true when the PERFORM statement is encountered, the specified procedures are not executed.

• In FORMAT 4, the values of identifiers and/or index-names are updated in an orderly fashion during the execution of the PERFORM statement.

References to identifier as the object of the VARYING, AFTER and FROM options also refers to index-name.

When an index-name is used, it is initialized and updated by the FORMAT 4 options of the PERFORM according to the rules of the SET statement.

• When one identifier is varied, the following sequence occurs:

Identifier-1 is set equal to its starting value (identifier-2 or literal-2).

If condition-1 is false, procedure-name-1 through procedure-name-2 is executed once.

The value of identifier-1 is updated by the increment or decrement value of identifier-3 (or literal-3) and condition-1 is retested.

The preceding two steps are repeated until condition-1 is true. At this point, control is transferred to the next executable statement following the PERFORM statement.

If condition-1 is true when the PERFORM statement is encountered, the procedures are not executed and control passes directly to the next executable statement after the PERFORM statement.

FIGURE 2-27 is a flowchart for the VARYING option of the PERFORM statement having one condition.

• When two identifiers are varied, the following sequence occurs.

Identifier-1 and identifier-4 are set to their respective initial values (identifier-2/literal-2 and identifier-5/literal-5).

Condition-1 is evaluated. If true, control is passed to the next executable statement following the PERFORM statement. If false, condition-2 is evaluated.

If condition-2 is false, procedure-name-1 through procedure-name-2 is executed.

2.4.9.26 PERFORM STATEMENT (Cont.)

Identifier-4 is updated by identifier-6 (or literal-6) and condition-2 is reevaluated.

If condition-2 is false, the preceding two steps are repeated.

If condition-2 is true, identifier-4 is set to its initial value (identifier-5).

Identifier-1 is updated by identifier-3 (literal-3).

The cycle of the preceding six steps is repeated until condition-1 is true.

If condition-1 was true when the PERFORM was encountered, identifier-1 and identifier-4 contain their initial values.

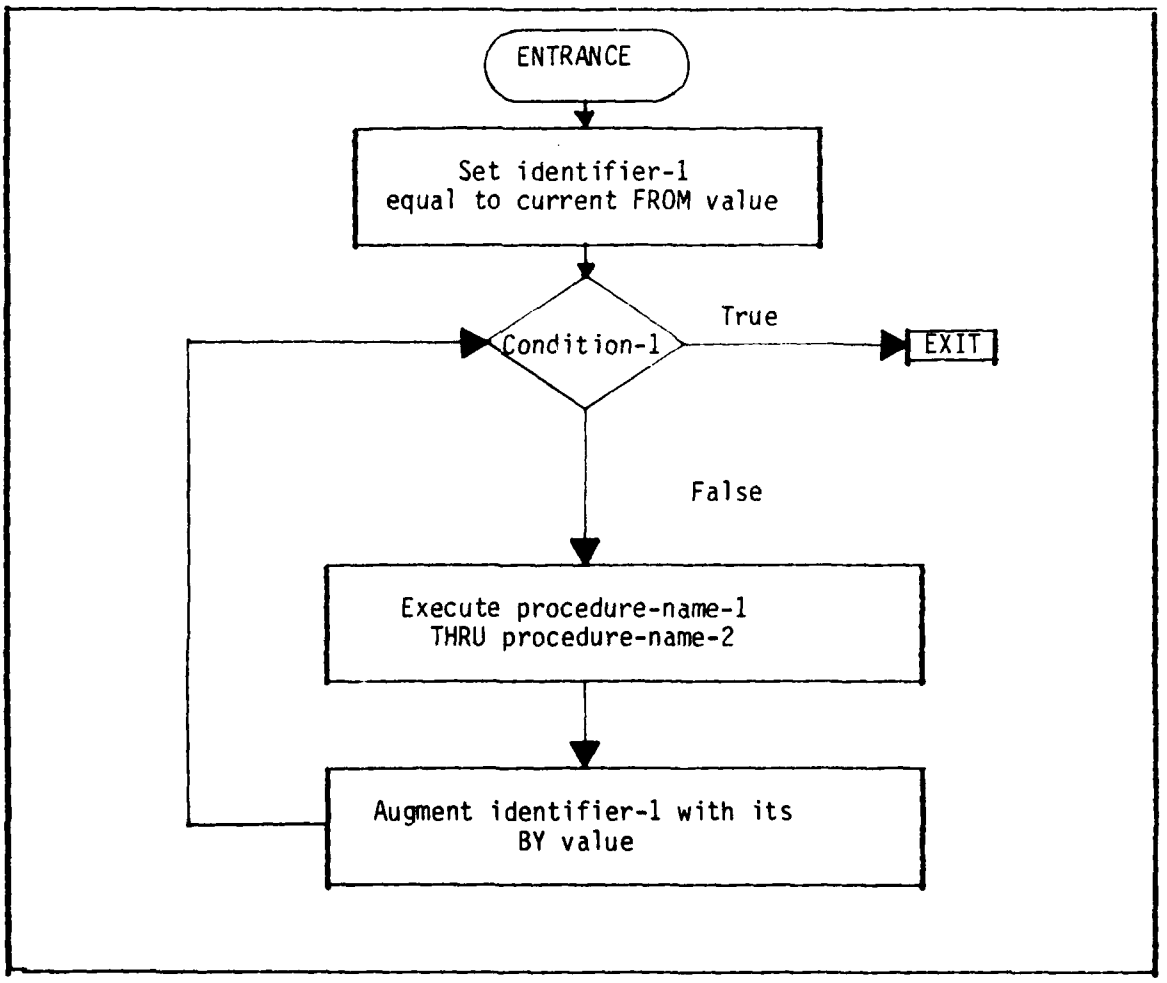



FIGURE 2-27

2.4.9.26 PERFORM STATEMENT. (Cont.)

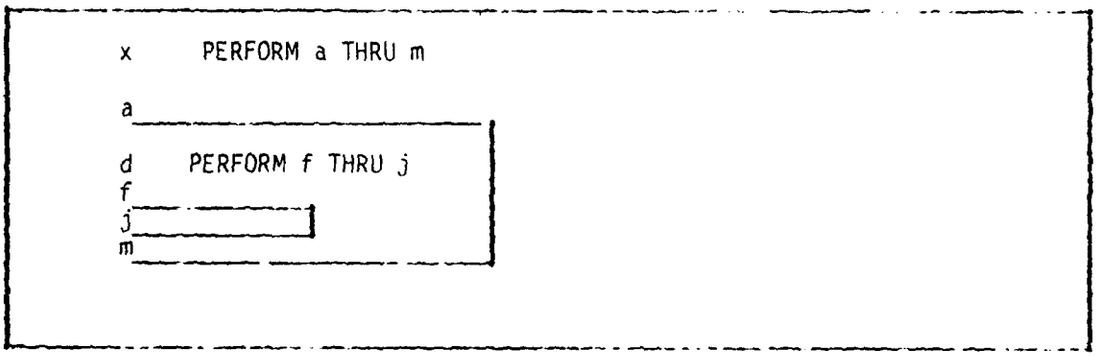
FIGURE 2-28 is a flowchart for the VARYING option of the PERFORM statement having two conditions.

- When three identifiers are varied, the following sequence occurs:

The mechanism is the same as for two identifiers except that identifier-7 goes through the complete cycle each time that identifier-4 is updated by identifier-6 (literal-6) which in turn goes through a complete cycle each time identifier-1 is varied.

FIGURE 2-29 is a flowchart for the VARYING option of the PERFORM statement having three conditions.

- If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement initiated within the range of another PERFORM statement must not allow control to pass to the exit of the first PERFORM statement. Two or more such active PERFORM statements may not have a common exit.



2.4.9.26 PERFORM STATEMENT. (Cont.)

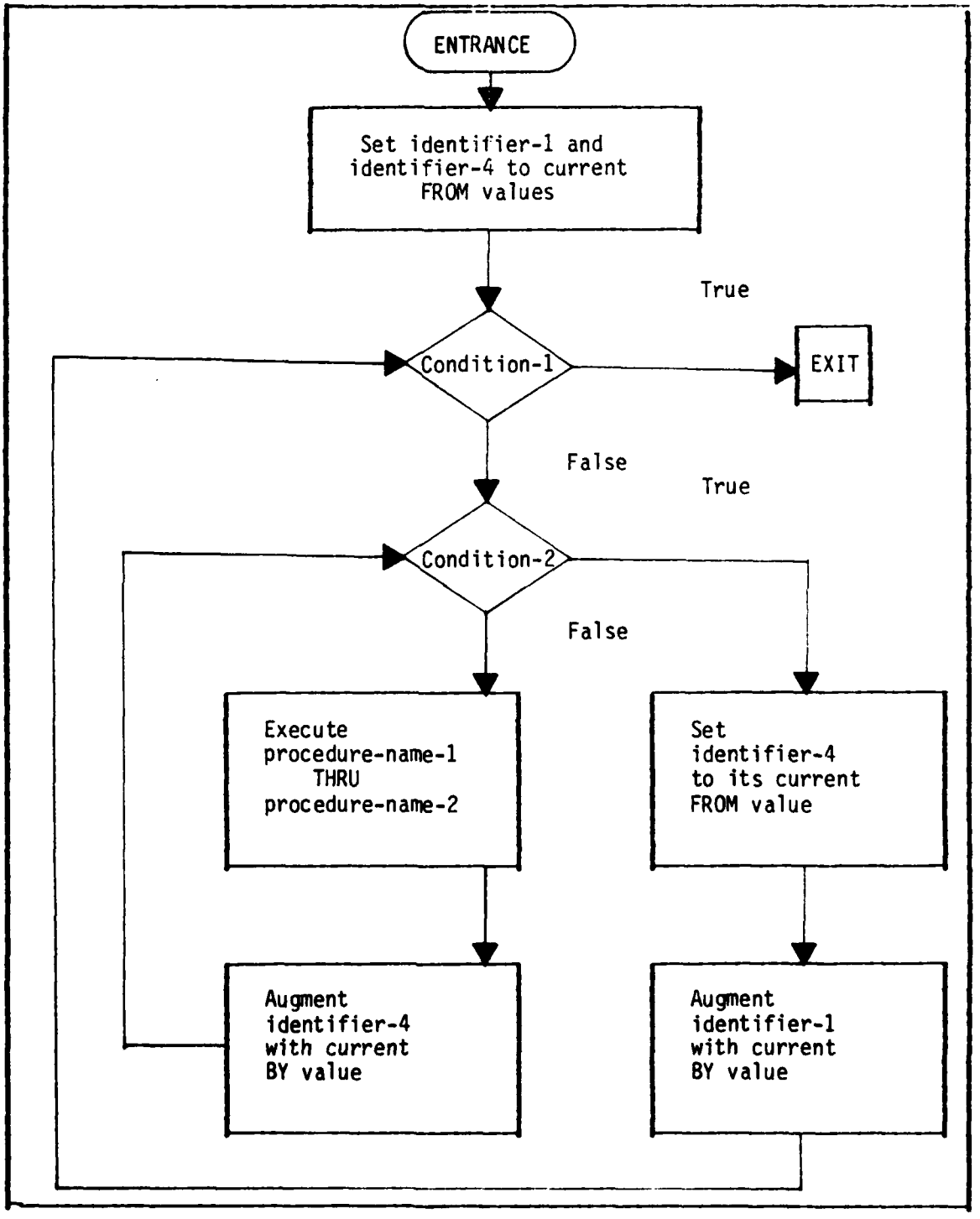



FIGURE 2-28

2.4.9.26 PERFORM STATEMENT. (Cont.)

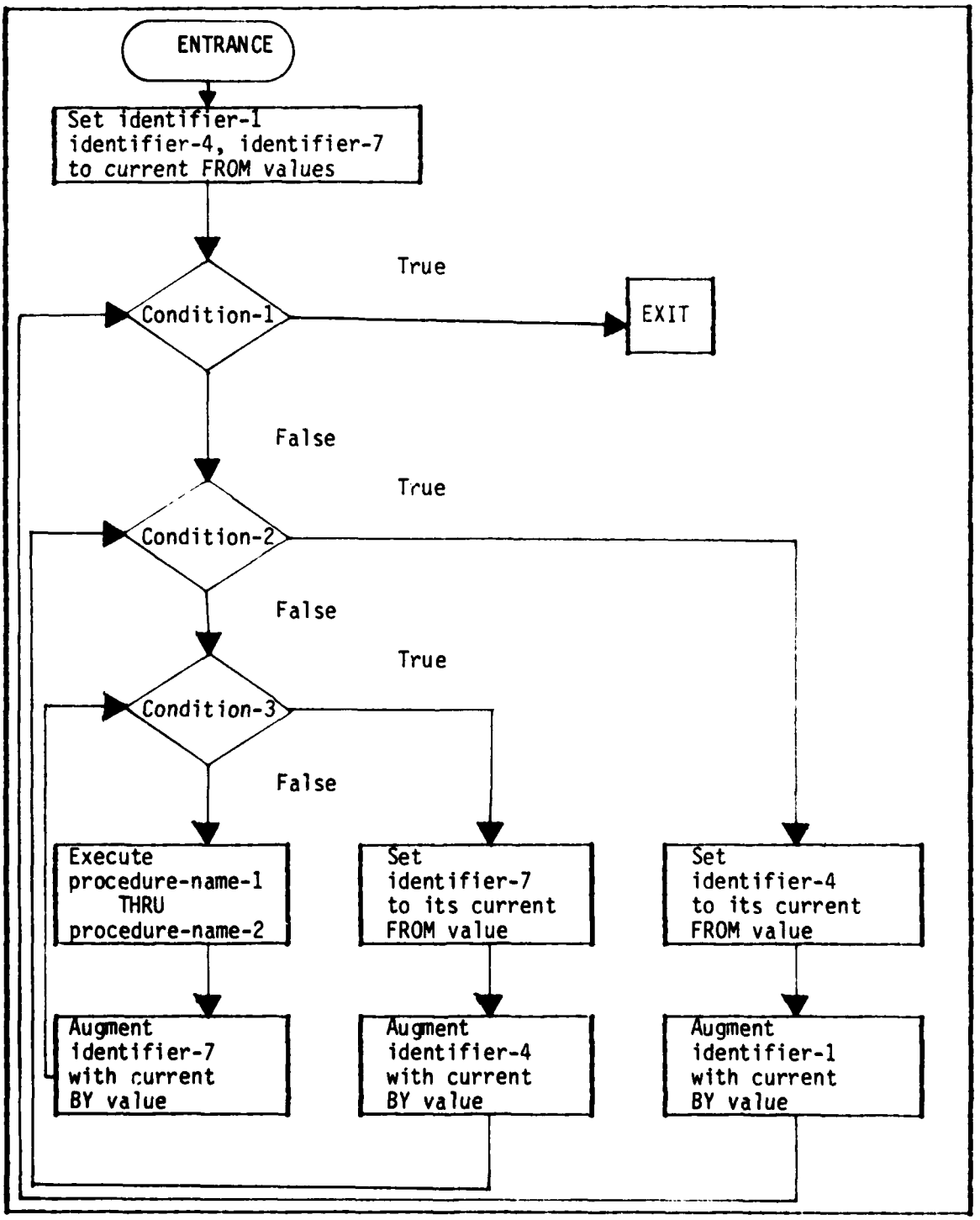
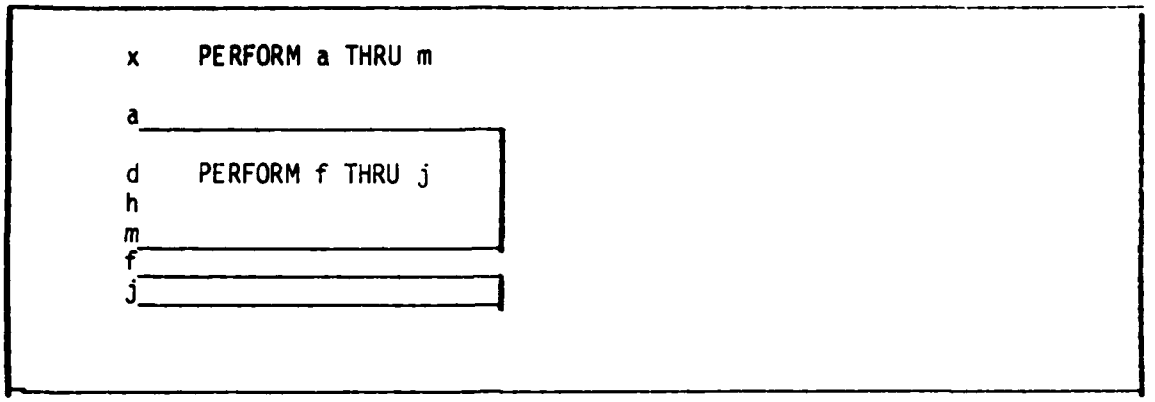
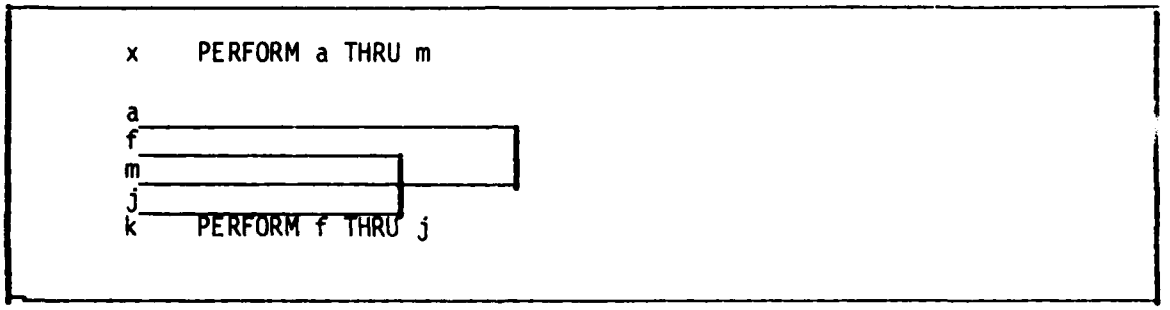


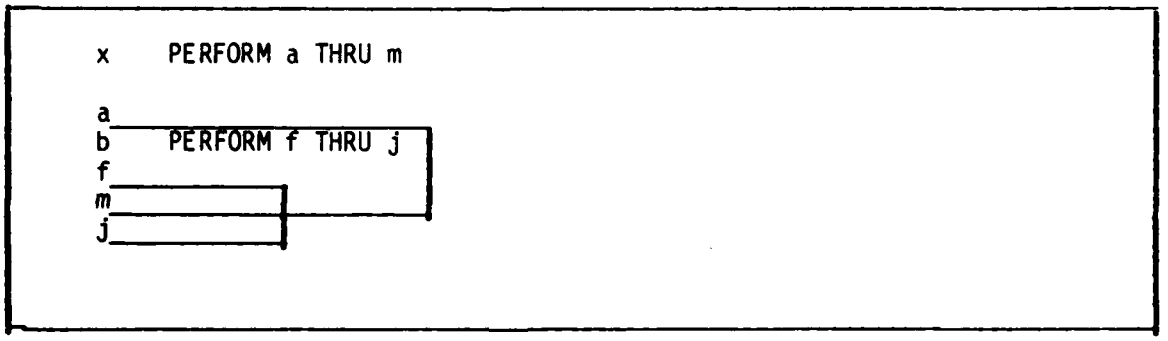

FIGURE 2-29

2.4.9.26 PERFORM STATEMENT. (Cont.)




• A PERFORM initiated within the range of an active PERFORM cannot allow control to pass to the exit of the first PERFORM statement.



2.4.9.26 PERFORM STATEMENT. (Cont.)

- Refer to RESTRICTION ON PERFORM STATEMENTS, COBOL SEGMENTATION FACILITY.

USACSC GUIDELINES.

- Groups of procedural statements that are frequently executed in an object program should be executed using the PERFORM mechanism. This technique reduces the size of the object program and also organizes the program in a way that can improve the program documentation and facilitate its maintenance. The grouping of input-output procedures into subroutines called by PERFORM statements is especially useful.

- Always execute the last statement of a series of routines being operated on by a PERFORM statement. When branching out of the routine, make sure control will eventually return to the last statement of the routine. This statement should be an EXIT statement. Although no code is generated, the EXIT statement allows a programer to immediately recognize the extent of a series of routines within the range of a PERFORM statement.

- Always either PERFORM with the THRU option, or PERFORM section-name. A PERFORM paragraph-name can cause trouble for the programer trying to maintain the program. For example, if a paragraph must be broken into two paragraphs, the programer must examine every statement to determine whether or not this paragraph is written the range of a PERFORM statement. Then all statements referencing the paragraph-name must be changed to PERFORM THRU statements.

- Only PERFORM A THRU A-EXIT, do not PERFORM A THRU C-EXIT when using structured programing techniques. PERFORM statement should never include a range of paragraphs.

2.4.9.27 READ STATEMENT.

FUNCTION. The READ statement makes available the next logical record from an input file.

FORMAT.

FORMAT 1.

```
READ file-name RECORD      [ INTO identifier ]

    { AT END     imperative-statement }
```

FORMAT 2.

```
READ file-name RECORD      [ INTO identifier ]

    { INVALID KEY     imperative-statement }
```

SYNTAX RULES.

- The INTO phrase must not be used when the input file contains variable-length records. The storage area associated with identifier and the record area associated with file-name must not be the same area.
- The AT END phrase must be specified for sequential files.
- For files in sequential access mode, FORMAT 1 must be used.
- For RANDOM or INDEXED-SEQUENTIAL files, FORMAT 2 is used.
- The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name.

GENERAL RULES.

- The file-name must be defined by a file description entry in the DATA DIVISION.

2.4.9.27 READ STATEMENT. (Cont.)

• An OPEN statement must be executed for the file prior to the first READ of that file. This positions the current record pointer to the beginning of the file.

• Execution of a READ statement causes the current record pointer to point to the next existing record in the file. This record is made available to the program in the input area defined by the associated record description entry.

• In FORMAT 2, the record the READ makes available is a function of the organization of the file and of the key fields associated with it.

• If a file contains more than one type of logical record, these records automatically share the same storage area by implicit redefinition of the Ø1-level record descriptions.

• Only information present in the current record is accessible. No reference can be made by a PROCEDURE DIVISION statement to information beyond the current record. Unpredictable results will result from referencing the nth occurrence of data that occurs fewer than n times.

• The record remains in the input area until the next READ for that file is executed.

• When the INTO option is specified, the following rules apply:

Identifier must be a WORKING-STORAGE SECTION entry, a LINKAGE SECTION entry, or an output record in the FILE SECTION. If identifier is an output record, its associated file must have been opened before the READ was executed.

This option acts as a combination READ statement and MOVE statement. The record is read then moved from the record area to the identifier area. Any subscripting or indexing associated with the identifier is evaluated after the record is read and immediately before it is moved into the area associated with the identifier.

Data is moved according to the COBOL rules for the MOVE statement.

The record being read is available in both the input record area and the area defined by identifier.

• In a multi-volume file, if the end of a volume (tape reel or mass storage unit) is detected during the execution of a READ statement, and the logical end of the file has not been reached, the following occurs:

The standard ending label procedures are executed.

A volume switch is executed.

The standard beginning label procedures are executed.

The first data record of the new volume is made available.

2.4.9.27 READ STATEMENT. (Cont.)

• When the last logical record in a file has been read, the next READ attempted for that file produces an end-of-file condition.

Control is passed to the imperative-statement associated with the AT END phrase.

The contents of the input record area is now unpredictable.

Another READ cannot be executed for that file without an intervening CLOSE statement followed by an OPEN statement for that file.

USACSC GUIDELINES.

• The number of READ statements should be minimized; normally only one per file is required. The technique to effect this is to code the READ as a separate procedure and to PERFORM it whenever the READ operation is required.

• For a file with several types of records (multiple Ø1 record descriptions) the type of record read should be determined immediately after the READ has been accomplished. Remember that only one logical record at a time is available (not one record of each type).

• After end-of-file has been encountered, the execution of statements that refer to the area record of the file must be avoided, because no record is available to refer to.

• Never execute a READ statement for a file on which end-of-file has been detected, unless it has been subsequently closed and reopened as input.

• The primary reason for using the READ INTO option of the READ statement is as follows:

Ease of debugging programing errors. It is far easier to determine which record in a file is being processed when that record is to be found in the WORKING-STORAGE of a program. Determining which record is the current one when processing in the buffers of a blocked file requires an extreme amount of expertise and is time-consuming.

Avoidance of system problems. The IBM OS MVT operating system takes away the program's buffers immediately after the at-end condition has been detected for any file. This forces the program to avoid all tests of data in the buffer areas. Better programing practice is the READ ... INTO a WORKING-STORAGE area, setting some condition in the area (such as moving HIGH-VALUES to the area) when the end-of-file condition is detected. The programer's only alternative in a situation where two files are being bounced together for comparison is to resort to use of switches or separate blocks of code specifically for the after-at-end conditions. Neither is normally desirable as a standard programing practice.

2.4.9.28 RELEASE STATEMENT.

FUNCTION. The RELEASE statement causes the records named by record-name to be released to the initial phase of a SORT operation.

FORMAT.

```
RELEASE record-name [FROM identifier]
```

SYNTAX RULES.

- Record-name must be the name of a logical record in the associated sort-file description (SD).
- Record-name and identifier must not refer to the same storage area.
- A RELEASE statement must only be used within the range of an input procedure associated with a SORT statement.

GENERAL RULES.

- The FROM option makes the RELEASE statement equivalent to the statement MOVE identifier TO record-name, followed by a RELEASE statement.
- After the execution of the RELEASE statement, the logical record is no longer available. However, if the FROM option is used, information stored in the identifier is available.

2.4.9.29 RETURN STATEMENT.

FUNCTION. The RETURN statement obtains individual records in sorted order from the final phase of the SORT operation.

```
RETURN file-name RECORD [ INTO identifier ]
  AT END imperative-statement
```

SYNTAX RULES

- The file-name must be the name given in the sort-file-description (SD) entry that describes the records that are to be sorted.

- The identifier must be the name of a WORKING-STORAGE area or the name of an output record area.

- A RETURN statement must only be used within the range of an output procedure associated with a SORT statement for file-name.

- The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the record area associated with file-name must not be the same storage area.

GENERAL RULES.

- When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the RETURN statement.

- The execution of the RETURN statement causes the next record, in the order specified by the keys listed in the SORT statement, to be made available for processing in the record areas associated with the sort file.

- The INTO OPTION has the same effect as the MOVE statement for alphanumeric items. The implied MOVE does not occur if there is an AT END condition. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to the data item.

- When the INTO phrase is used, the data is availabale in both the input record area and the data area associated with identifier.

- If no next logical record exists for the file at the time of the execution of a RETURN statement, the AT END condition occurs.

- The imperative-statement in the AT END phrase specifies the action to be taken when all the sorted records have been completed and have been obtained from the sorting operation. The contents of the record areas associated with the file when the AT END condition occurs are undefined. After the execution of the imperative-statement in the AT END phrase, no RETURN statement may be executed as part of the current output procedure.

2.4.9.30 REWRITE STATEMENT.

FUNCTION. The REWRITE statement logically replaces a record existing in a mass storage file.

FORMAT.

```
REWRITE record-name      [FROM identifier]

[;INVALID KEY imperative-statement]
```

SYNTAX RULES.

- Record-name and identifier must not refer to the same storage area.

- Record-name is the name of a logical record in the FILE SECTION of the DATA DIVISION and may be qualified.

- The INVALID KEY phrase must be specified in the REWRITE statement for files for which an appropriate USE procedure is not specified.

- The INVALID KEY phrase must not be specified for a REWRITE statement which references a file in sequential access mode.

- The INVALID KEY phrase must be specified in the REWRITE statement for files in the random access mode for which an appropriate USE procedure is not specified.

GENERAL RULES.

- The file associated with record-name must be a mass storage file and must be open in the I-O mode at the time this file is executed.

- The last input-output statement executed for the asssociated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The MSCS logically replaces the record that was accessed by the READ statement.

- The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.

- After the execution of the REWRITE statement, the logical record that was rewritten is no longer available in the record area.

- The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of:

```
MOVE identifier TO record-name
```

2.4.9.30 REWRITE STATEMENT. (Cont.)

followed by the execution of the same REWRITE statement without the FROM phrase. The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of the REWRITE statement.

- The current record pointer is not affected by the execution of a REWRITE statement.

- The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

- The REWRITE statement must be used with caution since the original record is not available after its execution. The user must provide some form of backup capability for files that are being rewritten.

- The INVALID KEY condition exists when:

  a. The access mode is sequential and the value contained in the prime record key data item of the record to be replaced is not equal to the value of the prime record key of the last record read from this file, or

  b. The value contained in the prime record key data item does not equal that of any record stored in the file.

- The updating operation does not take place and the data in the record area is unaffected.

- For a file accessed in random access mode, the MSCS logically replaces the record specified by the contents of the RELATIVE KEY data item associated with the file. If the file does not contain the record specified by the key, the INVALID KEY condition exists. The updating operation does not take place and the data in the record area is unaffected.

- For a file in the sequential access mode, the record to be replaced is specified by the value contained in the prime record key. When the REWRITE statement is executed the value contained in the prime record key data item of the record to be replaced must be equal to the value of the prime record key of the last record read from this file.

USACSC GUIDELINES. None.

2.4.9.31 SEARCH STATEMENT.

FUNCTION. The SEARCH statement is used to search a table for an element that satisfies a specified condition and to adjust the associated index-name to indicate that table element. Refer to "Table Handling Feature" in "Special Features" section.

2.4.9.32 SET STATEMENT.

FUNCTION. The SET statement establishes reference points for table handling operations by setting index-names to values associated with table elements. The SET statement is used for initializing index-name values before transferring values between index-names and other elementary data items. Refer to "Table Handling Feature" in "Special Features" section.

2.4.9.33 SORT STATEMENT.

FUNCTION. The SORT statement provides the information that is needed to control the sorting operation. This statement directs the sorting operation to obtain the records that are to be sorted from an INPUT PROCEDURE or the USING file, to SORT the records by keys in either ASCENDING and/or DESCENDING order and then to make the sorted records available to either an OUTPUT PROCEDURE or to a GIVING FILE.

FORMAT.

```
SORT file-name-1  ON { ASCENDING  }   KEY data-name-1  [, data-name-2] ...
                     { DESCENDING }

                [ ON { ASCENDING  }   KEY data-name-3  [, data-name-4] ... ] ...
                     { DESCENDING }

  { INPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
  {                                     { THRU    }                 }
  { USING file-name-2                                               }

  { OUTPUT PROCEDURE IS section-name-3 [ { THROUGH } section-name-4 ] }
  {                                      { THRU    }                 }
  { GIVING file-name-3                                               }
```

SYNTAX RULES.

• File-name-1 must be the name given on the sort-file description (SD) entry in the Data Division. This entry describes the entry to be sorted.

• Section-name-1 represents the name of an INPUT PROCEDURE. Section-name-3 represents the name of an OUTPUT PROCEDURE.

• File-name-2 and file-name-3 must be described in a file description entry, not in a sort-file description entry, in the Data Division. The actual size of the logical record(s) described for file-name-2 and file-name-3 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, it is the programer's responsibility to describe the corresponding records in such a manner so as to insure that all KEYS are physically located in the same position and have the same data-format in every logical record of the sort-file.

• Data-name-1, data-name-2, data-name-3, and data-name-4 are KEY data-names and are subject to the following rules:

a. The data items identified by KEY data-names must be described in records associated with file-name-1.

b. KEY data-names may be qualified.

2.4.9.33 SORT STATEMENT. (Cont.)

c. The data items identified by KEY data-names must not be variable length items nor may they name group items which contain variable length data items.

d. If file-name-1 has more than one record description, then the data items identified by KEY data-names need be described in only one of the record descriptions.

e. None of the data items identified by KEY data-names can be described by an entry which either contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.

- The words THRU and THROUGH are equivalent.

- SORT statements may appear anywhere except in the declaratives portion of the Procedure Division or in an INPUT or OUTPUT PROCEDURE associated with a SORT statement.

- No more than one file-name from a multiple file reel can appear in the SORT statement.

- At least one ASCENDING or one DESCENDING clause must be specified.

- Keys must be listed from left to right in order of decreasing significance.

- Either section-name-2 or section-name-4 is required if that procedure terminates in a section other than the one in which it started.

GENERAL RULES.

- The ASCENDING and DESCENDING options specify the sequence in which the records are to be sorted based on the sort keys.

The ASCENDING option indicates that the sorting sequence is from the lowest value of the key to the highest value.

The DESCENDING option indicates that the sorting sequence is from the highest value of the key to the lowest value.

- The INPUT PROCEDURE option is used to indicate that the programer has written an INPUT PROCEDURE to process records before they are sorted.

The INPUT PROCEDURE must consist of one or more sections written consecutively which do not form a part of any OUTPUT PROCEDURE.

The INPUT PROCEDURE must include at least one RELEASE statement.

There are three restrictions on the procedural statements within an INPUT PROCEDURE.

1. The INPUT PROCEDURE must not contain a SORT statement.

2.4.9.33 SORT STATEMENT. (Cont.)

2. The INPUT PROCEDURE must not contain any transfers of control to points outside the INPUT PROCEDURE except for the CALL statement.

3. The remainder of the PROCEDURE DIVISION must not contain any transfers of control to points inside the INPUT PROCEDURE.

Section-name-1 is the name of the INPUT PROCEDURE; section-name-2 is the name of the last section that contains the INPUT PROCEDURE.

At the end of the INPUT PROCEDURE, the compiler inserts a return mechanism and then the records that have been released to file-name-1 are sorted.

• When the USING option is specified, all the records in file-name-2 are transferred automatically to file-name-1. The actual size of the logical record described for file-name-2 must be equal to the actual size of the logical record described for file-name-1. It is the programer's responsibility to describe the corresponding records in such a manner so as to cause an equal number of character positions to be allocated for the corresponding records. File-name-2 must not be open at the execution of the SORT. For this option, the compiler will generate instructions to open, read, release, and close file-name-2 automatically.

• The OUTPUT PROCEDURE is specified in order that control be passed to a procedure after all the records are sorted.

The OUTPUT PROCEDURE must consist of sections written consecutively which do not form a part of any INPUT PROCEDURE.

The OUTPUT PROCEDURE must include at least one RETURN statement.

The same three restrictions that apply for the INPUT PROCEDURE also apply for the OUTPUT PROCEDURE.

Section-name-3 is the name of the OUTPUT PROCEDURE; section-name-4 is the name of the last section that contains the OUTPUT PROCEDURE.

At the end of the OUTPUT PROCEDURE, the return mechanism provides for termination of the SORT and control then goes to the next sentence following the SORT statement.

• When the GIVING option is specified, all sorted records in file-name-1 are automatically transferred to file-name-3. The actual size of the logical record described for file-name-3 must be equal to the actual size of the logical record described for file-name-1. It is the programer's responsibility to describe the corresponding records in such a manner so as to cause an equal number of character positions to be allocated for the corresponding records. File-name-3 must not be open at the execution of the SORT. For this option, the compiler will generate instructions to open, return, write and close file-name-3 automatically.

File-name-3 must be a standard sequential file.

2.4.9.33 SORT STATEMENT. (Cont.)

- The areas described by data-names following key must not overlap.

- Output order of records with keys of equal value is unspecified.

- Segmentation can be used in programs containing SORT statements subject to the following restrictions:

If a SORT statement appears in a section in the permanent resident segment, any INPUT PROCEDURE or OUTPUT PROCEDURE specified must appear totally within non-independent segments or wholly within a single segment.

If a SORT statement appears in an overlayable segment, any INPUT PROCEDURE or OUTPUT PROCEDURE specified must appear totally within the permanent resident segment or wholly within the same overlayable segment.

- Nested SORTs are not allowed.

VENDORS' GUIDELINES.

- IBM.

- A character in the EBCDIC collating sequence (used for alphabetic, alphanumeric, etc., data items) is interpreted as not being signed. For numeric data items, characters are collated algebraically (as being signed).

- A maximum of 12 keys may be specified.

- The total length of all the keys must not be greater than 256 bytes.

- All key fields must be located within the first 4,092 bytes of a logical record.

- An IBM extension to ANSI COBOL allows GO TO and PERFORM statements in the remainder of the PROCEDURE DIVISION to refer to procedure-names within the input or output procedure.

- RESERVED WORD DATA ITEMS. For the IBM-360/370, four reserved word data items are supplied by the compiler for use in controlling the Sort operation. These data items must not be defined by the programer in the DATA DIVISION.

The first three items may have control information transferred to them at object time if the user specifies them as the receiving fields of statements such as MOVE. The information must be passed before the SORT statement is executed. The items are initialized to zero by the compiler, but are not reset after a Sort procedure is executed.

- SORT-FILE-SIZE (IBM Extension) is the name of a binary data item whose PICTURE is S9(8). It is used for the estimated number of records in the file to

2.4.9.33 SORT STATEMENT. (Cont.)

be sorted. If SORT-FILE-SIZE is omitted, it is assumed that the file contains the maximum number of records that can be processed with the available core size and number of work units. In order to make more efficient use of both main and intermediate storage, the SORT-FILE-SIZE special register should be used whenever possible. It is realized that the exact number of records in a file may not be known, however, an estimate of the file size should be moved to this register. If the estimate exceeds the maximum, the estimate will be ignored.

EXAMPLE. An example of using the SORT-FILE-SIZE register is as follows:

1. If the number of records in your file is 1,ØØØ, you just simply state:

```
MOVE 1,ØØØ to SORT-FILE-SIZE.
```

2. This statement causes the value 1,ØØØ to be placed in a special register that is a binary data item with a PICTURE of S9(8).

• SORT-CORE-SIZE (IBM Extension) is the name of a binary data item whose PICTURE is S9(8). It is used to specify the number of character positions of core storage available to the sorting operation if it is different from the core size that the Sort would normally use.

• SORT-MODE-SIZE (IBM Extension) is the name of a binary data item whose PICTURE is S9(5). It is used for variable-length records. If the length of most records in the file is significantly different from the average record length performance is improved by specifying the most frequently occurring record length. If SORT-MODE-SIZE is omitted, the average length is assumed. For example, if records vary in length from 2Ø to 1ØØ bytes, but most records are 3Ø bytes long, the value 3Ø should be moved to SORT-MODE-SIZE. The maximum record length handled by the Sort is 32,ØØØ characters.

2.4.9.33 SORT STATEMENT. (Cont.)

The following, FIGURE 2-30, is an example of the SORT statement and other basic statements that are used to make up the SORT facility, (OS).

```
000025      IDENTIFICATION DIVISION.
000050      PROGRAM-ID. SAMPSORT.
000075          .
000100          .
000125          .
000150      ENVIRONMENT DIVISION.
000175      CONFIGURATION SECTION.
000200      SOURCE-COMPUTER.  IBM-360-G40.
000225      OBJECT-COMPUTER.  IBM-360-G40.
000250      INPUT-OUTPUT SECTION.
000275      FILE-CONTROL.
000300          SELECT INFD ASSIGN TO UT-S-INFILE.
000325          SELECT OUTED ASSIGN TO UT-S-OUTFILE.
000350          SELECT SORTSD ASSIGN TO UT-S-SORTFILE.
000375      DATA DIVISION.
000400      FILE SECTION.
000425      FD INFD
000450          LABEL RECORDS ARE STANDARD
000475          RECORDING MODE IS F
000480          BLOCK CONTAINS 0 RECORDS
000500          RECORD CONTAINS 80 CHARACTERS
000525          DATA RECORD IS INREC.
000550      01  INREC                                PIC X(80).
000575      FD  OUTFD
000600          LABEL RECORDS ARE STANDARD
000625          RECORDING MODE IS F
000640          BLOCK CONTAINS 0 RECORDS
000650          DATA RECORD IS OUTREC.
000700      01  OUTREC.
000725          05  CARR-CON                        PIC X.
000750          05  REC                             PIC X(80).
000775      SD  SORTSD
000800          LABEL RECORDS ARE STANDARD
000825          RECORDING MODE IS F
```

FIGURE 2-30

2.4.9.33 SORT STATEMENT. (Cont.)

```
000850     RECORD CONTAINS 30 CHARACTERS
000875     DATA RECORD IS SORTREC.
000900 01  SORTREC.
000925     05  KEY-1                    PIC X(10).
000950     05  FILLER                   PIC X(20).
000975     05  KEY-2                    PIC X(20).
001000     05  FILLER                   PIC X(30).
001025 PROCEDURE DIVISION.
001050 0010-SORTSEC.
001075     OPEN INPUT INFD.
001100     OPEN OUTPUT OUTFD.
001125     SORT SORTSD ON ASCENDING KEY KEY-1 KEY-2
001150         INPUT PROCEDURE IS INPROC SECTION
001175         OUTPUT PROCEDURE IS OUTPROC SECTION.
001180     IF SORT-RETURN IS NOT EQUAL TO 0
001185         GO TO 0080-ABEND.
001225 0020-INPROC SECTION.
001250 0030-INSEC.
001275     READ INFD AT END GO TO 0030-INPROCEND.
001300     MOVE INREC TO SORTREC.
001325     RELEASE SORTREC.
001350     GO TO INSEC.
001375 0030-INPROCEND.
001380     CLOSE INFD.
001390 0040-EXIT.
001400     EXIT.
001425 OUTPROC SECTION.
001450 0050-OUTSEC.
001475     RETURN SORTSD AT END GO TO 0060-CLOSE-FILE.
001500     MOVE SORTREC TO REC.
001525     MOVE SPACE TO CARR-CON.
001550     WRITE OUTREC.
001575     GO TO OUTSEC.
001600 0060-CLOSE-FILE.
001625     CLOSE OUTFD.
001630 0070-EXIT.
001675     EXIT.
001700 0080-ABEND.
001725     DISPLAY 'ABNORMAL END OF SORT'.
           (ABNORMAL TERMINATION PROCEDURE)
           .
           .
           .
```

FIGURE 2-30 (Cont.)

2.4.9.33 SORT STATEMENT. (Cont.)

USACSC GUIDELINES.

• Specify the minimum of primary and intermediate storage needed to execute the sort successfully.

• Use the ANSI COBOL SORT feature rather than the system utility sort program since it provides:

Flexibility in the allocation of storage, record and file size specification.

The ability to process records before and/or after the sort.

Protection from user errors in handling JCL job stream.

The benefits of standard COBOL documentation.

• The sort execution speed can be increased if records and files are carefully described, considering the following:

In formatting SORT records, specify as sort keys only the minimum amount of data necessary. If you specify more than one sort key field, try to make all keys contiguous and ordered from major key to minor key. If key fields are contiguous, group them under one group item and use the group item as the sort key. These actions will improve sort execution speed.

The definition of sort keys has certain restrictions. KEYS must be physically located in the same relative position of every logical record in the sort file. KEYS must not contain OCCURS clauses, nor appear after a variable portion of a record.

When blocking input and output files, block as many records as possible. Low blocking factors are the greatest single causes of poor execution speed.

Sort records should be as small as possible (the physical block must be 18 bytes or longer to avoid noise record errors). Use of an input procedure to extract unnecessary fields before each record is released to the sort should be employed whenever feasible.

In handling large files, the limiting factor is not primary storage or record size, but the amount of intermediate storage available. If a large file is to be sorted, it is best to divide it into several sorts and merge them together later. This reduces the possibility of abnormal program termination due to inadequate intermediate storage.

To better handle intermediate storage allocations, anticipate the minimum storage requirements and specify the amount to be used for sorting.

On IBM equipment, SORT-RETURN must be checked and if it is not equal to zero, abnormal termination procedures observed.

2.4.9.33 SORT STATEMENT. (Cont.)

Input and output procedures must be terminated with an EXIT statement.

The IBM extension to ANSI COBOL allowing GO TO and PERFORM statements in the remainder of the PROCEDURE DIVISION to refer to procedure names within the input or output procedure will not be used.

Use of the DISPLAY statement within a SORT INPUT or OUTPUT procedure should be used with extreme care. The SORT INPUT procedure uses a WRITE BEFORE ADVANCING which results in an overprint on the SYSOUT file if the program has previously DISPLAY'ed a message.

NOTE: For additional information see paragraph 2.5.3, SORT FEATURE.

2.4.9.34 START STATEMENT.

FUNCTION. The START statement provides a basis for logical positioning within an indexed or relative file, for subsequent sequential retrieval of records.

FORMAT.

```
START file-name INVALID KEY imperative-statement
```

SYNTAX RULES.

- File-name must be the name of an indexed or relative file.
- File-name must be the name of a file with sequential or dynamic access.
- File-name must not be the name of a sort or merge file.
- File-name must be defined by a file description entry in the DATA DIVISION.

GENERAL RULES.

- Normally, an indexed file in the sequential mode is processed sequentially from the first record to the last or until the file is closed. If processing is to begin at other than the first record, a START statement must be executed after the OPEN but before the first READ statement. Processing will then continue sequentially until a START statement or a CLOSE statement is executed or until the end-of-file is reached.
- If processing is to begin at the first record, a START statement is not required before the first READ.
- The contents of the NOMINAL KEY are used as the key value of the record at which processing is to begin. In this instance, this key value must be placed in the data-name specified by the NOMINAL KEY clause for this file before the START statement is issued.
- When the INVALID KEY option is specified, control is passed to the imperative-statement following INVALID KEY when the contents of the NOMINAL KEY field are invalid. The key is considered invalid when the record is not found in the file.

USACSC GUIDELINES. None.

2.4.9.35 STOP STATEMENT.

FUNCTION. The STOP statement causes a permanent or temporary suspension of the execution of the object program. Data may be displayed upon the CONSOLE but the use of SYSLST or SYSOUT is preferred.

FORMAT.

```
STOP { RUN     }
     { literal }
```

SYNTAX RULES.

- Both the STOP RUN and the STOP literal statement must be followed by a period.

- The literal may be numeric or non-numeric or may be a figurative constant, except ALL.

- If the literal is numeric, it must be an unsigned integer.

GENERAL RULES.

- When the STOP RUN statement is executed, processing of the problem program is immediately terminated and control is returned to the operating system.

- The STOP RUN statement must appear as the only, or the last, imperative statement in a series of statements within a paragraph as no further processing will occur after its execution.

- When the STOP literal statement is executed, the information in the literal portion of the statement is displayed on the system logical output device. The problem program is then placed in a temporary suspension of execution mode, and may be restarted by the operator as dictated by system specifications.

- The STOP literal statement may be placed at any point within the logical flow of a problem program. Once the program is restarted by the operator, execution will continue with the next sequential instructions.

VENDORS' GUIDELINES.

- IBM.

The statement STOP RUN should not be used in a subprogram. In this case, control will not return to the main program, but will go directly to the operating system and terminate the entire job.

2.4.9.35 STOP STATEMENT (Cont.)

The contents of the literal portion of the STOP literal statement are displayed on the system console, then the program goes into a temporary halt condition. The operator restarts the problem program by keying in an end-of-block indication for that program (EOB key). As no other information is entered by the operator, this is the most efficient and least error-prone method of effecting a temporary halt in a user program control.

USACSC GUIDELINES.

- There should be only one STOP RUN statement in a program, and it should be the last entry in the program's logical flow. Ideally, it should be coded as a single sentence paragraph. The program would then either fall through into it in the case of a completely normal processing run or GO TO it in the case of an other than normal processing run. If there are certain tasks that must always be executed before the RUN STOP can be executed, those tasks may also be included in the same paragraph, placed before the STOP RUN statement. An example is a file CLOSE statement that must always be executed. It should be coded only once, and the logical place for it to be placed is in the paragraph with the STOP RUN.

- All open files must be closed before the STOP RUN statement is executed. OS will close all open files itself, but does so in a very inefficient fashion, after much time delay. DOS will not close files left open, causing probable error conditions in later processing of those files.

- Due to its greater efficiency and lesser chance for error, the STOP literal statement should always be used if possible in preference to the combination of a DISPLAY ... UPON CONSOLE followed by ACCEPT ... FROM CONSOLE to effect a program-controlled temporary halt in processing. Program halts of either kind, though, should be avoided whenever another method can be employed to achieve the same effect.

2.4.9.36 SUBTRACT STATEMENT.

FUNCTION. The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from one or more items, and set the values of one or more items equal to the results.

FORMAT.

FORMAT 1.

```
SUBTRACT { identifier-1 } [, identifier-2 ] ...
         { literal-1    } [, literal-2    ]
  FROM identifier-m [ ROUNDED ]
  [; ON SIZE ERROR imperative-statement]
```

FORMAT 2.

```
SUBTRACT { identifier-1 } [, identifier-2 ] ...
         { literal-1    } [, literal-2    ]
  FROM { identifier-m } GIVING identifier-n [ ROUNDED ]
       { literal-m    }
  [; ON SIZE ERROR imperative-statement]
```

SYNTAX RULES.

- Each identifier must refer to an elementary numeric data item except the object of the GIVING option (identifier-n) which may be a numeric-edited data item.

- Each literal must be a numeric literal.

- The maximum size of each operand is 18 digits. The maximum size of the result, after decimal alignment, is 18 digits.

GENERAL RULES.

- In FORMAT 1, all identifiers or literals preceding the word FROM are added together, and this total is subtracted from identifier-m, (identifier-n, etc.). The result of the subtraction is stored as the new value of identifier-m (identifier-n, etc.).

2.4.9.36 SUBTRACT STATEMENT. (Cont.)

• In FORMAT 2, all identifiers or literals preceding the word FROM are added together, and this total is subtracted from identifier-m. The result of the subtraction is stored in the object of the GIVING option, identifier-n.

• The GIVING, ROUNDED, and SIZE ERROR options are explained in Arithmetic Operations - PROCEDURE DIVISION.

VENDORS' GUIDELINES.

• IBM.

In FORMAT 2, only one operand is permitted as the object of the GIVING clause.

USACSC GUIDELINES.

• For efficient execution and ease of maintenance, SUBTRACT statements should be as simple as possible.

Do not use ON SIZE ERROR unnecessarily. This option increases execution time and takes more space whether a size error exists or not.

Do not use ROUNDED unnecessarily, for the same reasons as above. To ADD 5 and then MOVE to drop insignificant digits is more efficient.

Avoid using more than three operands before the word FROM.

• In regard to the data items specified as operands in arithmetic statements:

When they are all defined with the same USAGE, the relatively expensive operation of conversion is avoided. This does not apply to display items in the IBM-360/370. In these computers, display items must be converted to packed decimal before they can be used arithmetically.

When, for ADD and SUBTRACT, they are all defined with the same number of decimal places, the relatively expensive operation of scaling is avoided.

• In regard to a result item specified in the GIVING clause:

When it is defined with sufficient integer places to provide for the maximum integer value possible, the need for the ON SIZE ERROR clause is eliminated.

When it is defined with the same number of decimal places as the calculated result, the relatively expensive operation of truncation, rounding, or scaling (whichever applies in the particular case) is avoided.

2.4.9.36 SUBTRACT STATEMENT. (Cont.)

When it is defined with the same USAGE as the calculated result, a conversion operation is avoided.

For the IBM-360/370, when it is defined as signed, the operation of removing the sign (which is generated automatically by the hardware) is avoided.

2.4.9.37 USE STATEMENT.

FUNCTION.

• The USE statement is part of the DECLARATIVE Section which provides a method of including procedures that are invoked nonsynchronously; that is, they are executed not as part of the sequential coding but rather when a condition occurs which cannot normally be tested by the programer.

• The USE sentence specifies the procedure to be followed if an input/output error occurs during file processing.

FORMAT.

```
USE AFTER STANDARD ERROR PROCEDURE ON { file-name-1 } [file-name-] ... .
                                      { INPUT     }
                                      { OUTPUT    }
                                      { I-O       }
```

SYNTAX RULES.

• A USE statement, when present, must immediately follow a section header in the DECLARATIVES Section and must be followed by a period followed by a space. The remainder of the section must consist of zero, one or more procedural paragraphs that define the procedures to be used.

• The USE statement itself is never executed; it merely defines the conditions calling for the execution of the USE procedure. Recursive USE procedures are prohibited.

• No file-name may reference a sort or merge file.

GENERAL RULES.

• The designated procedures are executed by the input-output system at the appropriate time after completing the standard input-output error routine, or upon recognition of the INVALID KEY or AT END conditions, when the INVALID KEY phrase or AT END phrase, respectively, has not been specified in the input-output statement.

For relative or indexed files the INVALID KEY phrase is required only when no applicable USE AFTER ERROR procedure is specified.

• After execution of a USE procedure, control is returned to the invoking routine.

• Within a USE procedure, there must not be any reference to any non-declarative procedures. Conversely, in the non-declarative portion there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE statement having the procedures associated with such a USE statement.

2.4.9.37 USE STATEMENT. (Cont.)

• In the statements of the out-of-line procedures, the execution of a PERFORM statement affects that particular processing cycle only; that is, any other asynchronous processing cycle in the out-of-line set of procedural statements is not affected by another concurrent processing cycle in which a PERFORM statement is executed.

USACSC GUIDELINES.

• The programer may wish to consider using the USE statement as an assist to debugging in printing subtotals or conditions in the program processing at the point of ABEND.

NOTE: See paragraph 2.4.9.10 for USE FOR DEBUGGING STATEMENT.

2.4.9.38 WRITE STATEMENT.

FUNCTION. The WRITE statement releases a logical record for a standard sequential output file or input-output file. It can also be used for vertical positioning of lines within a logical page.

FORMAT.

FORMAT 1.

```
WRITE  record-name  [FROM  identifier-1]
[ [ { BEFORE } ADVANCING { { identifier-2 } [ LINE ] } ] ]
    [ { AFTER  }         { { integer      } [ LINES ] } ]
                         { { mnemonic-name }           }
                         { { PAGE          }           }
```

FORMAT 2.

```
WRITE  record-name [ FROM  identifier-1 ]
       [ INVALID KEY  imperative-statement ]
```

SYNTAX RULES.

- Record-name and identifier-1 must not reference the same storage area.
- The record-name is the name of a logical record in the FILE SECTION of the DATA DIVISION and may be qualified.
- Identifier-2 must be the name of an elementary integer data item.
- Integer or the value of identifier-2 may be zero.
- FORMAT 2 cannot be used for sequential files. See USE Statement.

GENERAL RULES.

- The associated file must be open in the OUTPUT or I-O mode at the time of the execution of this statement.
- After the WRITE statement is executed, the logical record released is no longer available in the record area unless the execution of the WRITE statement is unsuccessful due to an INVALID KEY condition or due to a boundary violation.
- The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of:

2.4.9.38 WRITE STATEMENT. (Cont.)

a. The statement:

MOVE identifier-1 TO record-name

according to the rules specified for the MOVE statement, followed by:

b. The same WRITE statement without the FROM phrase.

The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

After execution of the WRITE statement is complete, the information in the area referenced by identifier-1 is available, even though the information in the area referenced by record-name may not be.

- The current record pointer is unaffected by the execution of a WRITE statement.

- The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

- The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

- The number of character positions on a mass storage device required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.

- The execution of the WRITE statement releases a logical record to the operating system.

- The ADVANCING phrase allows control of the vertical positioning of each line on a representation of a printed page. If the ADVANCING phrase is not used, automatic advancing will be provided by the implementor to act as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

a. If integer is specified, the representation of the printed page is advanced the number of lines equal to the value of integer.

b. If the BEFORE ADVANCING phrase is used, the line is presented before the representation of the printed page is advanced according to rule a above.

c. If the AFTER ADVANCING phrase is used, the line is presented after the representation of the printed page is advanced according tc rule a above.

d. If PAGE is specified, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page. If the record to be written is associated with a file whose file description entry does not contain a LINAGE clause, the repositioning to the next logical page is accomplished in accordance with an implementor-defined

2.4.9.38 WRITE STATEMENT. (Cont.)

technique. If page has no meaning in conjunction with a specific device, then advancing will be provided by the implementor to act as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING 1 LINE.

- When an attempt is made to write beyond the externally defined boundaries of a sequential file, an exception condition exists and the contents of the record area are unaffected. The following action takes place:

a. The value of the FILE STATUS data item, if any, of the associated file is set to a value indicating a boundary violation.

b. If a USE AFTER STANDARD EXCEPTION declarative is explicitly or implicitly specified for the file, that declarative procedure will then be executed.

c. If a USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for the file, the result is undefined.

- If an end of volume (tape reel or disk unit) occurs during a WRITE statement for that file, the following occurs:

The standard ending label procedure is executed.

A volume switch is executed.

The standard beginning volume procedure is executed.

- If the ADVANCING clause is omitted, the default value is BEFORE.

- Execution of the WRITE statement causes the contents of the record area to be released. The MSCS utilizes the content of the record keys in such a way that subsequent access of the record key may be made based upon any of those specified record keys.

- The value of the prime record key must be unique within the records in the file.

- The data item specified as the prime record key must be set by the program to the desired value prior to the execution of the WRITE statement.

- When a file is opened in the output mode, records may be placed into the file by one of the following:

a. If the access mode is sequential, the WRITE statement will cause a record to be released to the MSCS. The first record will have a relative record number of one (1) and subsequent records released will have relative record numbers of 2, 3, 4, ... . If the RELATIVE KEY data item has been specified in the file control entry for the associated file, the relative record number of the record just released will be placed into the RELATIVE KEY data item by the MSCS during execution of the WRITE statement.

2.4.9.38 WRITE STATEMENT. (Cont.)

b. If the access mode is random or dynamic, prior to the execution of the WRITE statement the value of the RELATIVE KEY data item must be initialized in the program with the relative record number to be associated with the record in the record area. That record is then released to the MSCS by exection of the WRITE statement.

• When a file is open in the I-O mode and the access mode is random or dynamic, records are to be inserted in the associated file. The value of the RELATIVE KEY data item must be initialized by the program with the relative record number to be associated with the record in the record area. Execution of a WRITE statement then causes the contents of the record area to be released to the MSCS.

• The INVALID KEY condition exists under the following circumstances:

a. When sequential access mode is specified for a file opened in the output mode, and the value of the prime record key is not greater than the value of the prime record key of the previous record, or

b. When the file is opened in the output or I-O mode, and the value of the prime record key is equal to the value of a prime record key of a record already existing in the file, or

c. When the access mode is random or dynamic, and the RELATIVE KEY data item specifies a record which already exists in the file, or

d. When an attempt is made to write beyond the externally defined boundaries of the file.

• When the INVALID KEY condition is recognized the execution of the WRITE statement is unsuccessful, the contents of the record area are unaffected and the FILE STATUS data item, if any, associated with file-name of the associated file is set to a value indicating the cause of the condition.

VENDORS' GUIDELINES.

• IBM.

The integer must be a positive integer of less than 100.

In the ADVANCING option, the first character of each record in a file must be reserved by the user for the control character. The compiler generates instructions to insert the appropriate carriage control character as the first character in the record. It is the user's responsibility to see that the carriage control tape is correctly punched.

Certain statements (DISPLAY, EXHIBIT, WRITE AFTER ADVANCING) cause the printer to space before printing. A simple WRITE or a WRITE BEFORE ADVANCING causes the printer to space after printing. Mixing these statements may cause overprinting.

The BEFORE option is significantly more efficient (in terms of execution time) than the AFTER option.

2.4.9.38 WRITE STATEMENT. (Cont.)

USACSC GUIDELINES.

- The number of WRITE statements should be minimized; normally one per record type is adequate. The technique to effect this is to code each WRITE in a separate procedure and PERFORM it whenever the WRITE operation is required.

- When a file contains records of several types it is often best to define the records in WORKING-STORAGE, build them there and move them to the record area just prior to the WRITE. The Øls following the FD would then be effectively "dummy" record descriptions required only to indicate the various record lengths that can be written. This technique avoids the problem of a record not being available in the output area after a WRITE has been executed. In this way, data common to each record can be saved for use in more than one record without having to move it separately to the record each time.

## 2.5 SPECIAL FEATURES.

### 2.5.1 STRUCTURED PROGRAMING STATEMENTS - MetaCOBOL Macro Facility.

To enhance the COBOL language and facilitate the implementation of structured programming, additional statements are available through MetaCOBOL. They are described below.

2.5.1.1 DO STATEMENT.

FUNCTION. The DO Statement causes the execution of the module specified by procedure-name.

FORMAT.

```
DO procedure-name [ description ]
```

SYNTAX RULES.

- The procedure-name in the DO statement should match the procedure-name specified in the Identification Section of another module.

- The description, if present, serves as documentation commentary only. The description should be brief. For example:

```
DO READ-MASTER (ACQUIRE NEXT MASTER)
```

- "READ-MASTER" is the name of another module. The function of that module is to "ACQUIRE NEXT MASTER" record.

GENERAL RULES.

For any DO where the procedure-name does not match a named Identification Section, dummy paragraphs will be generated at the end of the program to make it compilable and executable.

USACSC GUIDELINES. None.

2.5.1.2 DO WHILE STATEMENT.

FUNCTION. DO WHILE causes execution of the statement repeatedly in a loop as long as the condition is true.

FORMAT.

```
DO WHILE      condition
      statement ...
ENDDO
```

SYNTAX RULES.

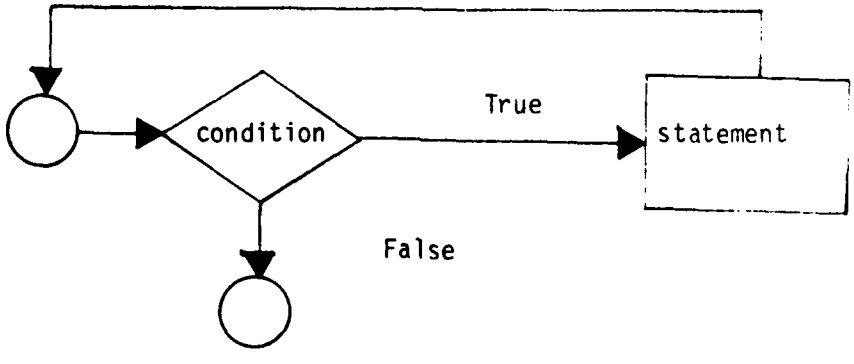
- The condition can be any valid COBOL conditional test, including compound relationships connected by AND or OR. The statement can be one or more COBOL imperative statements or special structured programing statements, including DO WHILE.

For example:

```
DO WHILE B1-VALID-FLAG NOT = SPACE
    DO READ-TRANS (ACQUIRE NEXT TRANS)
    IF NOT M-EOF
        DO TRANS-VALIDITY-CHECK (CHECK SEQUENCE)
    ENDIF
ENDDO
```

GENERAL RULES.

- The ENDDO signals the end of the list of statements forming the loop represented by statement for the previous unterminated DO WHILE.



2.5.1.2 DO WHILE STATEMENT. (Cont.)

- *If B1-VALID-FLAG is non-blank, the DO and IF are executed. This process* is repeated until B1-VALID-FLAG is blank, presumably as a result of executions of module READ-TRANS. The processing continues at the next statement after ENDDO.

USACSC GUIDELINES. None.

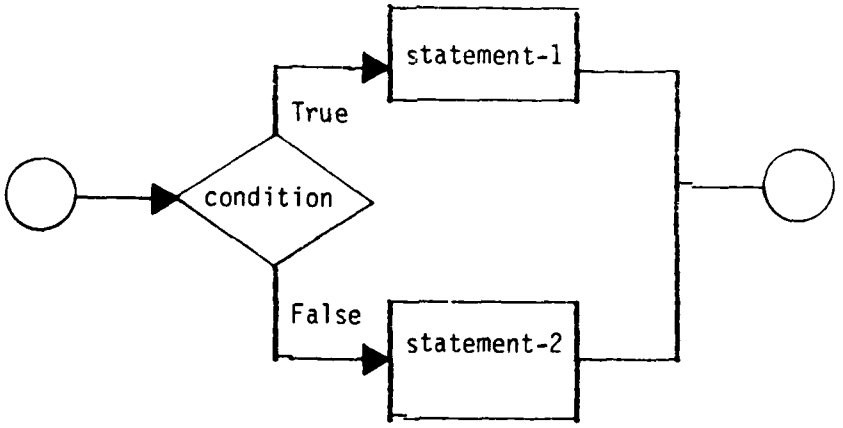
2.5.1.3 IF STATEMENT.

FUNCTION. This IF statement is similar to a COBOL IF except that it is terminated by ENDIF and not a period.

FORMAT.

```
IF          condition
             statement-1 ...
ELSE         statement-2 ...
ENDIF
```

SYNTAX RULES. The ENDIF terminates the previous unterminated IF only.




The condition can be any valid COBOL conditional test, including compound conditionals using AND or OR. Both statement-1 and statement-2 can be one or more COBOL imperative statements or special structured programing statements, including IF.

Statement-1 is executed if the condition is true, and statement-2 is executed if the condition is not true.

For example:

```
IF M-FLAG = 'S'
    MOVE M-SAVE TO W-MASTER
ELSE
    PERFORM READ-MASTER (ACQUIRE NEXT MASTER)
ENDIF
```

2.5.1.3 IF STATEMENT. (Cont.)

GENERAL RULES. Either the MOVE or PERFORM is executed, depending on whether M-FLAG contains an 'S'.

USACSC GUIDELINES. None.

2.5.1.4 DO UNTIL STATEMENT.

FUNCTION. The DO UNTIL statement causes execution of one or more imperative statements over again and again until a specified condition is met.

FORMAT.

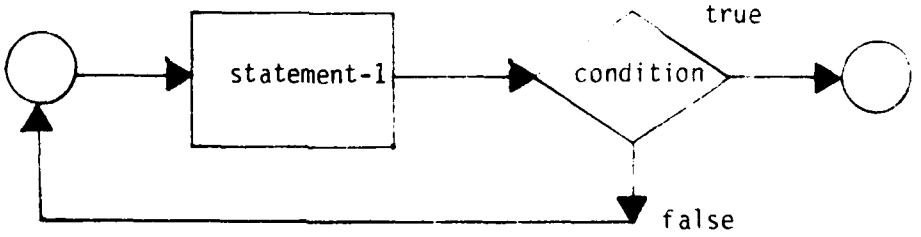
```
DO UNTIL   condition
           statement-1

ENDDO
```

SYNTAX RULES.

The condition can be any valid COBOL conditional test, including compound conditionals connected by AND or OR. Statement-1 can be one or more COBOL imperative statements or special structured programming statements including DO UNTIL.

GENERAL RULES.

- DO UNTIL always causes execution of statement-1 and then causes execution of statement-1 repeatedly in a loop as long as the condition is false. The ENDDO signals the end of the list of statements forming the loop represented by statement-1 and statement-2 for the previous unterminated LOOP.




- For example:

```
DO UNTIL TRANS-KEY IS GREATER THAN T-OLD-KEY
    PERFORM NEXT-TRANS
ENDDO
```

The module NEXT-TRANS will be executed and then if TRANS-KEY is greater than T-OLD-KEY the construct will be exited otherwise the processing returns to perform the module NEXT-TRANS again.

USACSC GUIDELINES. None.

2.5.1.5 CASE STATEMENT.

FUNCTION. The CASE statement causes control to be passed to one or more imperative statements based on the value of an integer variable.

FORMAT.

```
CASENTRY        identifier
CASE            condition-1 (or condition ...)
                    statement-1
[ CASE          condition-2
                    statement-2 ]
...
[ CASE          condition-n
                    statement-n ]
ELSECASE            statement-n+1
ENDCASE
```

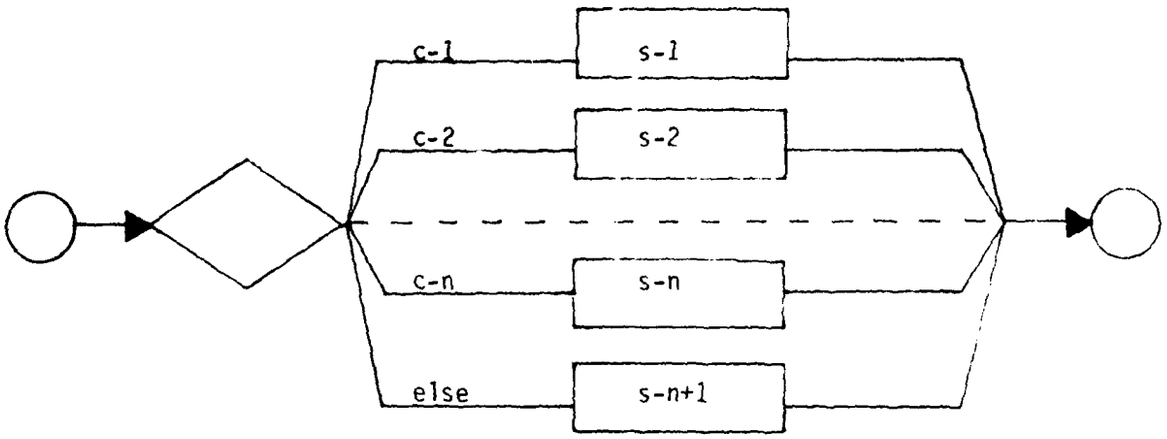
SYNTAX RULES.

- The identifier is a field whose value is to be checked against n literals and/or fields for an equal condition.
- Any statement can be one or more COBOL imperative statements or special structured programming statements, including up to 9 nested CASE statements.

GENERAL RULES.

- The CASE statement is similar to the IF structured programming statement except that it provides an n-way decision instead of a binary decision.

2.5.1.5 CASE STATEMENT. (Cont.)




• The CASE statement for structured COBOL uses the GO TO ... DEPENDING ON statement. This verb permits the programer to select one of a set of procedures depending upon the value of an integer whose range is from 1 to the number of procedure names listed in the statement. For any integer outside these limits the GO TO statement is ignored and control passes to the statement which follows it.

• Statement-n is executed if the corresponding condition-n is true.

For example:

```
CASENTRY TRANSACTION-CODE
CASE          'A'
              DO ADD-ROUTINE (PROCESS ADD TRANSACTION)
CASE          'C'
              DO CHANGE-ROUTINE (PROCESS CHANGE TRANSACTION)
CASE          'D'
              DO DELETE-ROUTINE (PROCESS DELETE TRANSACTION)
ELSECASE      DO ERROR-T-ROUTINE (TRANSACTION CODE ERROR)
ENDCASE
```

• The field TRANSACTION-CODE can contain three valid values, each of which specifies further processing. Any other value in this field is an error.

• Case 'A' could as well have been 'A' or 'B' or 'Z' should more than one case require the same processing.

• The literal may be numeric or any legal length alphabetic literal.

• The use of variable names is prohibited from use as this violates the "spirit" of structured programming in that such a use does not lend itself to code which is easily understood and is not likely to be a part of an eventual COBOL version.

2.5.2 COBOL SEGMENTATION FACILITY. The COBOL segmentation facility provides a means by which the user may communicate with the compiler to specify object program overlay requirements. The segmentation feature permits segmentation of procedures only. Sections of a program are resident or overlayable in core according to user-specified section numbers. In this way, a large program can be executed in a defined area of core storage by limiting the number of segments that are in core storage at any one time.

2.5.2.1 Organization of Segmentation Facility. When segmentation is used, the entire PROCEDURE DIVISION must be written in sections. Each section must be classified as belonging either to a fixed portion or to one of the independent segments of the object program. Segmentation in no way affects the need for qualification of procedure-names to ensure uniqueness.

- The fixed portion is defined as that part of the object program which is logically treated as if it were always in memory. This portion of the program is composed of two types of segments: fixed permanent segments and fixed overlayable segments.

A fixed permanent segment is a segment in the fixed portion which cannot be overlaid by any other part of the program. These segments must be available for reference at all times.

A fixed overlayable segment is a segment in the fixed portion which, although logically treated as if it were always in storage, can be overlaid (if necessary) by another segment to optimize storage utilization. Such a segment, if called for by the program, is always made available in the state it was when it was last used. These segments are generally less frequently used than the fixed permanent segments.

- The independent segment portion is made up of independent segments. An independent segment is a part of the object program which can overlay, or be overlaid by, either a fixed overlayable segment or another independent segment. These segments are not as frequently used in the program logic as the fixed portion segments. An independent segment is always considered to be in its initial state each time it is made available to the program.

2.5.2.2 Segment Classification. Sections to be segmented are classified using a system of segment priority numbers. The numbers are assigned using the following criteria:

- Logic Requirements. Sections which must be available for reference at all times, or which are referred to very frequently, are normally classified as belonging to one of the fixed permanent segments. Sections which are less frequently used are normally classified as belonging to one of the fixed overlayable segments or to one of the independent segments, depending on logic requirements.

- Frequency of use. Generally, the more frequently a section is referred to, the lower its segment-number; the less frequently it is used, the higher its segment-number.

- Relationship to other sections. Sections which frequently communicate with one another should be given the same priority number. All sections with the same priority number make up a single program segment.

2.5.2.3 Segmentation Control. The logical sequence of a source program is the same as the physical sequence except for specific transfers of control. When the sections of a segment are not contiguous, the object module is reordered to make the sections contiguous. However, the compiler provides transfers to assure that the program logic flow is followed as written. The compiler also inserts instructions to load and/or initialize a segment when necessary. Control may be transferred within a source program to any paragraph in a section; it is not mandatory to transfer control to the beginning of a section.

2.5.2.4 Structure Of Program Segments.

- Priority-numbers. Sections are combined into segments by a system of priority numbers. The priority number is included in the section header. All sections with the same priority number constitute a segment.

FORMAT.

| | | |
|---|---|---|
| section-name | SECTION | priority-number. |

RULES.

- The priority-number must be an integer from Ø through 99.

Segments with priority-numbers from Ø through 49 are part of the fixed portion of the object program, unless altered by the SEGMENT-LIMIT clause.

Segments with priority-numbers from 5Ø through 99 are independent segments.

2.5.2.4 Structure Of Program Segments. (Cont.)

If a priority-number is omitted from a section header, the priority is assumed to be zero.

• Segment-limit. Normally, all program segments with priority numbers from Ø to 49 are treated as permanent segments. However, at times the permanent segments plus the largest overlayable segment exceed core limitations. In this case, it is necessary to decrease the number of permanent segments. This is done by using the SEGMENT-LIMIT clause in the OBJECT-COMPUTER paragraph.

FORMAT.

```
[ SEGMENT-LIMIT   IS priority-number ]
```

RULES.

• Priority-number must be an integer from Ø to 49.

• When the SEGMENT-LIMIT clause is used, only those segments with priority-numbers up to (but not including) the designated segment limit are considered as permanent segments.

• The segments having priority numbers from the segment limit through 49 are considered as overlayable fixed segments.

• If the SEGMENT-LIMIT clause is omitted, all segments having priority numbers from Ø through 49 are considered to be permanent segments.

2.5.2.5 Restrictions On PERFORM Statement.

• A PERFORM statement that appears in a section whose priority number is lower than the segment limit can only refer to the following:

Sections with priority numbers lower than 5Ø (fixed or fixed overlayable segments).

Sections wholly contained in a single segment whose priority number is higher than 49 (an independent segment).

• A PERFORM statement that appears in a section whose priority number is equal to or higher than the segment limit can only refer to the following:

Sections within the same priority number as the section containing the PERFORM statement (entirely contained in one independent segment).

2.5.2.5 Restrictions On PERFORM Statement. (Cont.)

Sections with priority numbers that are lower than the segment limit (fixed permanent segments).

- When a procedure-name in a permanent segment is referred to by a PERFORM statement in an independent segment, the independent segment is reinitialized upon exit from the performed procedures.

2.5.2.6 Example of Segmentation. Refer to FIGURE 2-31.

```
IDENTIFICATION DIVISION.
.
.
.
ENVIRONMENT DIVISION.
OBJECT-COMPUTER.  IBM-360-H50.
  SEGMENT-LIMIT   IS 10.
.
.
.
DATA DIVISION.
.
.
.
PROCEDURE DIVISION.
.
.
.
0010-NBR-1 SECTION 5.
.
.
0020-NBR-2 SECTION 5.
.
.
.
0030-NBR-3 SECTION 18.
.
.
.
0040-NBR-4 SECTION 5.
.
.
.
0050-NBR-5 SECTION 53.
.
.
.
```

FIGURE 2-31

2.5.2.6 Example of Segmentation. (Cont.)

```
ØØ6Ø-NBR-6 SECTION 18.
.
.
.
ØØ7Ø-NBR-7 SECTION 53.
.
.
.
```

FIGURE 2-31 (Cont.)

• In the preceding sample program, shown at FIGURE 2-31, seven sections composing three segments are illustrated. Sections 1, 2, and 4 have a priority number of 5 which is less than the segment-limit. They constitute a fixed permanent segment which will always be available in core. Sections 3 and 6 have a priority number of 18 which is greater than the segment-limit but less than 49. They constitute a fixed overlayable segment which will normally remain in core but may be overlaid if necessary. This segment is always in its last used state whenever it is returned to core. Sections 5 and 7 have a priority number of 53 which places this segment in the category of an independent segment which is in its original state each time it is called into core.

2.5.2.7 USACSC Guidelines For Segmentation.

• *If a program is to be segmented, a primary requirement is that the* possibility be considered during the system and program design phase. During this phase the nature and sequence of data to be processed should be investigated to determine paths of the program which will be required all of the time and those with lower percentage of use. A great amount of preplanning is required to determine efficient overlay structure.

• The decision to segment a program is a tradeoff of the memory to be saved against the time lost reading and reloading segment blocks. Some considerations in creating a segmented program are:

The amount of memory to be saved.

The percentage of use of a procedure. A procedure that is used more than 5% of the time should probably be a permanent segment. A procedure used only 5% of the time will be required on an average once in every twenty records. This could amount to reloading the segment 5ØØ times if 1Ø,ØØØ transactions are processed.

ANSI segmentation is best utilized where successive execution of overlayable segments are possible. "Flip-flop" processing of overlayable segments is not recommended because it requires excessive CPU time.

2.5.2.7 USACSC Guidelines For Segmentation. (Cont.)

The size of a segmented program is equal to the size of the fixed segment plus the largest overlayable or independent segment. The size of a segmented program with COBOL sort input/output procedures is equal to the fixed segment plus the largest overlayable segment plus the largest overlayable sort segment.

For techniques on debugging an OS segmented program, see OS COBOL Program Debugging Aids in the Debugging Aids section of this manual.

Segmentation of very large programs can also be accomplished through separately compiled and called overlays. Refer to USACSCM Executive Software Catalogs (18-2-B series) (P54ATP DOS, P52ATU OS) for additional information.

2.5.3 SORT FEATURE.

2.5.3.1 Introduction.

• Sorting is the process of rearranging a group of records into a new sequence. A COBOL sort-file is the group of records to be rearranged and the sort keys are fields within the sort-file records according to which the records are sequenced. Records may be sorted in conjunction with regular file processing which may consist of addition, deletion, creation, editing, validation, or any other modification either to the records being sorted or to other records being processed in the sort program. This processing may occur before and/or after the sorting occurs. The procedures that cause special processing before sorting are called input procedures. Those that cause processing after sorting are called output procedures. For example, a transaction file can be validated, sorted, and then applied to a master file all in a single program. Or, a master file can be updated, sorted into a different sequence for printing, then edited and written in the new sequence.

• The basic elements of the COBOL Sort Feature are the SORT statement in the PROCEDURE Division and the Sort-File-Description (SD) entry, with its associated record description entries, in the Data Division. A sorting operation is based on sort-keys named in the SORT statement. Sort-keys are defined in the record description associated with the SD entry. The sort-key data items may be evaluated in ascending or descending order, or in a mixture of the two; that is, the sort-keys may be specified as ascending or descending, independent of one another, and the sequence of the sorted records will conform to the mixture specified.

• The SORT statement in the PROCEDURE DIVISION is the primary element of a source program that includes a COBOL sorting operation. The term sorting operation is used to mean not only the manipulation by the sort program of sort files on the basis of the sort-keys designated by the COBOL programer, but also to include the method of making records available to, and retrieving records from the sort-work files. A sort-work file is the collection of records actually involved in the sorting operation as they exist on an intermediate device. To use the Sort Feature, the COBOL programer must provide information related to sorting in the Environment, Data, and Procedure Divisions of the source program. The specifications for these language elements follow this general description.

2.5.3.2 Environment Division Sort Feature.

• In the Environment Division, the programer must write a SELECT entry for the sort-file in addition to the SELECT entries required for all the files used as input and output in the program. The SELECT entry for the sort-file specifies, in the ASSIGN clause, the hardware device that the sort operation uses for the work files generated during the execution of the sort.

FORMAT.

• In the ENVIRONMENT DIVISION, the following format must be used for IBM:

```
SELECT                file-name

ASSIGN TO    [integer-1]    system-name-1    [system-name-2] ...
```

RULES.

• File-name identifies the sort-file to the compiler.

• The ASSIGN clause is used to describe the sort work files. See language element - ASSIGN clause for DOS/OS specifications for integer-1 and system-name-n.

• The I-O-CONTROL paragraph specifies the memory area to be shared by different sort-files, if applicable. The SAME SORT AREA clause specifies that two or more files, at least one of which is a sort-file, will use the same memory area for processing of the current logical record. The logical record of only one of the files can exist in the record area at one time. During the execution of a SORT statement that refers to a sort-file named in this clause, any non-sort-files named in the clause must not be opened.

2.5.3.3 Data Division Sort Feature.

• In the Data Division, the programer must include File Description entries (FD) for all files that are used to provide input to or receive output from the sort program. He must also write a Sort-File Description (SD) entry and its associated record description entries to describe the records that are to be sorted, including their sort-key fields.

FORMAT.

• The sort-file-description entry must appear in the File Section. The following format must be used:

2.5.3.3 Data Division Sort Feature. (Cont.)

```
SD                file-name

    [;RECORD CONTAINS   [ integer-1  TO ]  integer-2 CHARACTERS ]

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

RULES.

- File-name is the name used to describe the records to be sorted.

- The RECORD CONTAINS clause merely specifies the size of the data records. The actual size and mode of the records is determined from the level-Ø1 descriptions associated with a given SD entry.

2.5.3.4 Procedure Division Sort Feature.

- In the Procedure Division, the programer specifies in the SORT statement the sort file-name, the sort-key-names, and whether the records are to have special processing. If there is to be such processing, the Procedure Division must also include the program sections that perform the processing.

*NOTE: For additional information see paragraph 2.4.9.33, SORT STATEMENT.*

2.5.4 TABLE HANDLING FEATURE.

2.5.4.1 Introduction.

TABLES OF DATA. Tables of data are common components of business data processing problems.

- Tables composed of contiguous data items are defined in COBOL by including the OCCURS clause in their data description entries. The clause specifies that the item is to be repeated as many times as stated. *The item* is considered to be a table element and its name and description apply to each repetition or occurrence. Since each occurrence of a table element does not have assigned to it a unique data-name, reference to a desired occurrence may be made only by specifying the data-name of the table element together with the occurrence number of the desired table element. The occurrence number is known as a subscript, and this technique of specifying individual table elements is called subscripting.

2.5.4.1 Introduction. (Cont.)

• In order to facilitate such operations as table searching and manipulating specific items, a technique called *indexing* is also available.

INITIAL VALUES OF TABLES. In the WORKING-STORAGE SECTION, initial values of elements within tables are specified in one of the following ways:

• The table may be described as a record by a set of contiguous data description entries, each of which specifies the VALUE of an element, or part of an element, of the table. In defining the record and its elements, any data description clause (USAGE, PICTURE, etc.) may be used to complete the definitions where required. This form is required when the elements of the table require separate handling due to synchronization, USAGE, etc. The hierarchical structure of the table is then shown by use of the REDEFINES entry and its associated subordinate entries. The subordinate entries, following the REDEFINES entry which are repeated due to OCCURS clauses, must not contain VALUE clauses.

• When the elements of a table do not require separate handling, the value of the entire table may be given in the entry defining the entire table. The lower level entries will show the hierarchical structure of the table; lower level entries must not contain VALUE clauses.

2.5.4.2 Subscripting.

FORMAT.

```
data-name     (subscript[, subscript] ...)
```

• Subscripts are inclosed in parentheses following the space after data-name, which is the table element.

• When more than one subscript is used, a comma must separate the subscripts, and a space must follow each comma.

• No space must appear between the inclosing parentheses and their adjacent subscripts.

SUBSCRIPTING. A method by which occurrence numbers may be specified is to append one or more subscripts to the data-name. A subscript is an integer greater than zero, whose value specifies the occurrence number of an element within the group item that has the next lower level-number. The subscript can be represented either by a literal which is an integer or by a data-name which is defined elsewhere as a numeric elementary item with no character positions to the right of the assumed decimal point. In either case, the subscript inclosed in parentheses, is written immediately following the name of the table element. A table element must include as many subscripts as there are dimensions in the table whose element is being referred to. That is, there must be a subscript for each OCCURS clause in the hierarchy containing the data-name including the data-name itself.

2.5.4.2 Subscripting. (Cont.)

EXAMPLE OF SUBSCRIPTING. For a table with three levels of subscripting, the following Data Division entries would result in a storage layout as shown below.

```
Ø1  PARTY-TABLE REDEFINES TABLE.
    Ø5  PARTY-CODE OCCURS 3 TIMES.
        1Ø  AGE-CODE OCCURS 3 TIMES.
            15 M-F-INFO OCCURS 2 TIMES PICTURE S9(7)V9
               USAGE DISPLAY.
```

• Reference to elementary items within PARTY-TABLE is made by use of a name that is subscripted. A typical Procedure Division statement might be:

```
MOVE M-F-INFO (PARTY, AGE, M-F) TO M-F-RECORD.
```

• In order to use the Table Handling feature, the programer must provice certain information in the Data Division and Procedure Division of the program. Refer to FIGURE 2-32 for Storage Layout for Party-Table.

• When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the data organization. If a multi-dimensional table is thought of as a series of nested tables, the most inclusive or outermost table is considered to be the major table, with the innermost or least inclusive table being the minor table, then the subscripts are written from left to right in the order major, intermediate, and minor.

2.5.4.2 Subscripting. (Cont.)

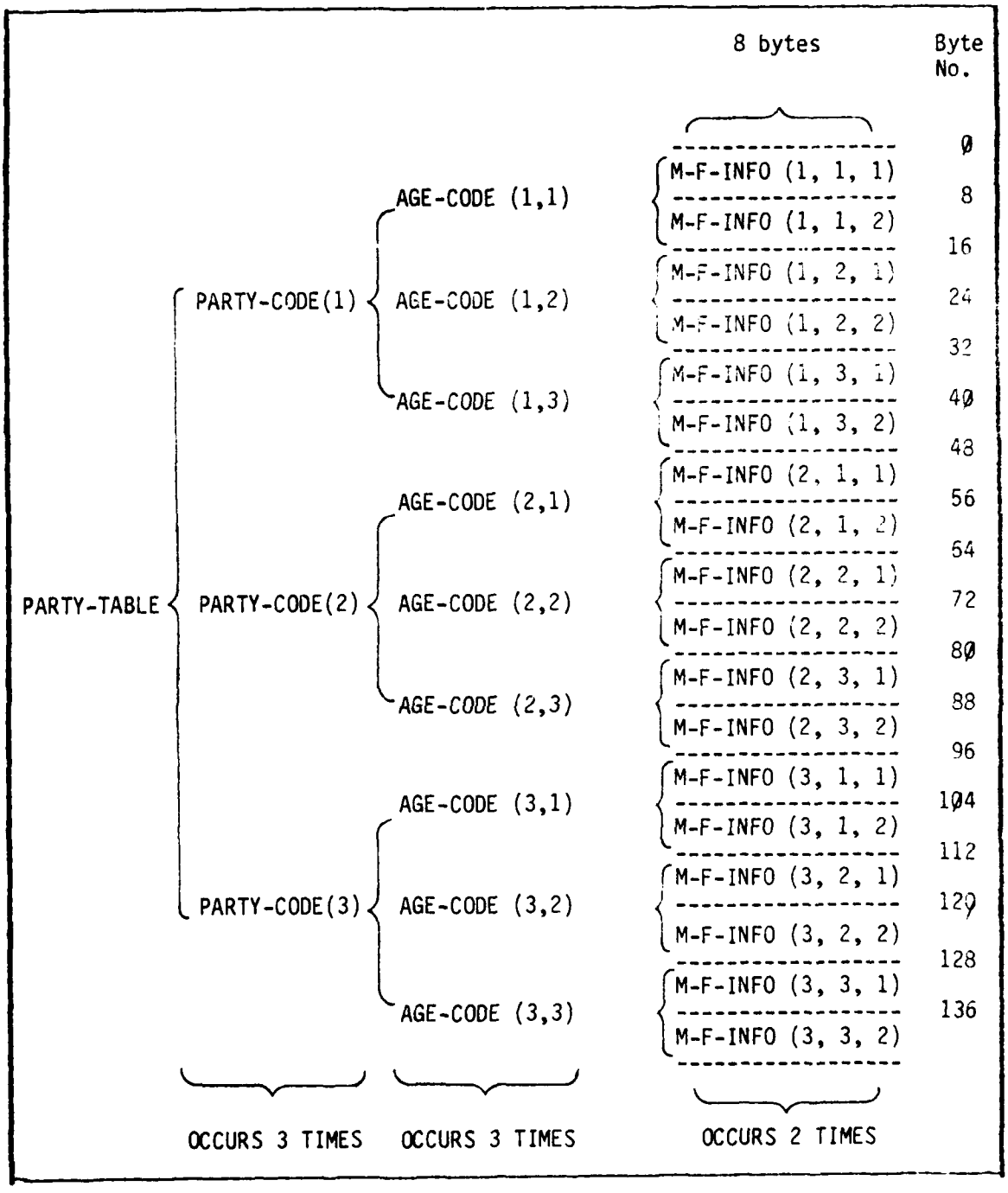



FIGURE 2-32

2.5.4.2 Subscripting. (Cont.)

- A reference to an item must not be subscripted if the item is not a table element or an item or condition name within a table element.

- The lowest permissible subscript value is 1. The highest permissible subscript value in any particular case is the maximum number of occurrences of the item as expressed in the OCCURS clause.

- When a data-name is used as a subscript, it may be used to refer to items within many different tables. These tables need not have elements of the same size. The data-name may also appear as the only subscript with one item and as one of two or three subscripts with another item. Also, it is permissible to mix literal and data-name subscripts.

For example:

```
ARGMNT (12, KEY, 2).
```

2.5.4.3 OCCURS STATEMENT.

• Another method of referring to items in a table is indexing. To use this technique the programer assigns one or more index-names to an item whose data description contains an OCCURS clause. An index is assigned to a given level of a table by using the INDEXED BY clause in the definition of the table.

FORMAT.

• The following are formats of the OCCURS clause showing how an index-name is assigned through use of the INDEXED BY clause.

FORMAT 1.

```
OCCURS integer-2 TIMES
    [ { ASCENDING  } KEY IS data-name-2 [data-name-3] ... ] ...
      { DESCENDING }
    [INDEXED BY index-name-1 [index-name-2] ... ]
```

FORMAT 2.

```
OCCURS integer-1 TO integer-2 DEPENDING ON data-name-1
    [ { ASCENDING  } KEY IS data-name-2 [data-name-3]... ] ...
      { DESCENDING }
    [ INDEXED BY index-name-1 [ index-name-2] ...]
```

At object time the contents of the index-name will correspond to an occurrence number for that specific dimension of the table to which the index-name was assigned.

• The OCCURS clause may not be specified in a data description entry that:

Has a level-Ø1 or level-77 number.

Describes an item whose size is variable (the number of times the item may occur can be variable).

• Data-name-1, the object of the DEPENDING option has the following restrictions:

Must be described as a positive integer.

Must not exceed integer-2 in value.

2.5.4.3 OCCURS STATEMENT. (Cont.)

Must not be subscripted.

Must not appear in the variable portion of the record.

• Any Data Division entry which contains an OCCURS DEPENDING clause or which has a subordinate entry which contains a DEPENDING clause, cannot be the object of a REDEFINES clause.

KEY OPTION: The KEY option is used in conjunction with the INDEXED BY option in the execution of a SEARCH ALL statement. The KEY option is used to indicate that the repeated data is arranged in ASCENDING or in DESCENDING order, according to the values contained in data-name-2, data-name-3, etc.

• Data-name-2 must be either the name of the entry containing an OCCURS clause, or it must be an entry subordinate to the entry containing the OCCURS clause. If data-name-3 is the subject of this table entry, it is the only key that may be specified for this table. If data-name-3 is not the subject of this table entry, all the keys identified by data-name-2, data-name-3, etc.:

Must be subordinate to the subject of the table entry itself.

Must not be subordinate to any other entry that contains an OCCURS clause.

Must not themselves contain an OCCURS clause.

• When the KEY option is specified, the following rules apply:

Keys must be listed in decreasing order of significance.

The total number of keys for a given table element must not exceed 12.

The sum of the lengths of all the keys associated with one table element must not exceed 256.

A key may have the following usages: DISPLAY or COMPUTATIONAL.

2.5.4.4 Indexing.

DIRECT INDEXING. The INDEXED BY option of the OCCURS clause refers to data-names accessed by indexing. Direct indexing is specified when a reference is made to a table element, and the item is followed by its related index-name.

- The index-name(s) is not defined elsewhere in the program, since its allocation and format are dependent on the system, and, not being data, cannot be associated with any data hierarchy.

- The number of index-names for a Data Division entry must not exceed twelve.

- An index-name must be initialized through a SET statement before it is used.

- Each index-name is a fullword in length and contains a binary value that represents an actual displacement from the beginning of the table that corresponds to an occurrence number in the table. The value is calculated as the occurrence number minus one, multiplied by the length of the entry that is indexed by this index-name.

EXAMPLE.

For example, if the programer writes

```
A OCCURS 15 TIMES INDEXED BY Z PICTURE IS X(1Ø).
```

on the fifth occurrence of 'A', the binary value contained in 'Z' will be:

```
Z = (5 - 1) * 1Ø = 4Ø
```

- Note that, for a table entry of variable length, the value contained in the index-name entry will become invalid when the table entry length is changed, unless the user issues a new SET statement to correct the value contained in the index-name.

2.5.4.4 Indexing. (Cont.)

FORMAT.

• The following format is used to imply that a data-name belongs to a structure with three nested levels of OCCURS.

```
data-name      (index-1 [, index-2]  ...)
```

• Data-name is originally defined in an OCCURS clause as part of the record description entries in the Data Division. The OCCURS clause must also use the INDEXED BY option.

• Index-1, index-2, index-3 must correspond to the index-names assigned in the INDEXED BY clause in the Data Division.

RELATIVE INDEXING. Relative indexing is specified when the index-name is followed by one of the operators, + or -, and a numeric literal. The numeric literal is considered to be an occurrence number and is converted to an index value before being added to, or subtracted from the corresponding index-name.

FORMAT.

• The following format is used to imply that a data-name belongs to a structure with three nested levels of OCCURS.

```
data-name (index-name { + } integer
                      { - }
[ , index-name-2 { - } integer ]  [ , index-name-3 { + } integer ] )
                 { + }                             { - }
```

• Data-name is originally defined in an OCCURS clause as part of the record description entries in the Data Division. The OCCURS clause must also use the INDEXED BY option and the index-names used in the Data Division must correspond to those in the Procedure Division. Integer must be a numeric literal.

INDEX DATA ITEM. An index data item is an elementary item (not necessarily connected with any table) that can be used to save index-name values for future reference. An index data item must be assigned an index-name value (i.e., (occurrence number -1)* entry length) through the SET statement. Such a value corresponds to an occurrence number in a table. The USAGE IS INDEX clause allows the programer to specify index data items.

2.5.4.4 Indexing. (Cont.)

FORMAT.

```
77 INDEX-ITEM [USAGE is ] INDEX.
```

• The USAGE IS INDEX clause may be written at any level. If a group item is described with the USAGE IS INDEX clause, it is the elementary items within the group that are index data items; the group itself is not an index data item, and the group name cannot be used in SEARCH and SET statements or in relation conditions. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

• An index data item can be referred to directly only in a SEARCH or SET statement or in a relation condition. An index data item can be part of a group which is referred to in a MOVE or an input/output statement. When such operations are executed, however, there is no conversion of the contents of the index data item.

• An index data item cannot be a conditional variable.

• The SYNCHRONIZED, JUSTIFIED, PICTURE, BLANK WHEN ZERO, or VALUE clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

2.5.4.5 Procedure Division Considerations for Table Handling.

• The SEARCH and the SET statements may be used to facilitate table handling. In addition, there are special rules involving table handling elements when they are used in relation conditions.

RELATION CONDITIONS. Comparisons involving index-names and/or index data items conform to the following rules:

• The comparison of two index-names is actually the comparison of the corresponding occurrence numbers.

• In the comparison of an index-name with a data item (other than an index data item), or in the comparison of an index-name with a literal, the occurrence number that corresponds to the value of the index-name is compared with the data item or literal.

• In the comparison of an index data item with an index-name or another index data item, the actual values are compared without conversion.

• Any other comparison involving an index data item is illegal.

2.5.4.5 Procedure Division Considerations for Table Handling. (Cont.)

TABLE LOOK-UP.

• SEARCH STATEMENT. The SEARCH statement is used to search a table for an element that satisfies a specified condition, and to adjust the value of the associated index-name to the occurrence number corresponding to that table element.

FORMAT.

FORMAT 1.

```
SEARCH identifier-1  [VARYING { index-name-1 } ]
                              { identifier-2 }

  [ AT END imperative-statement-1]

    WHEN condition-1  { imperative-statement-2 }
                      { NEXT SENTENCE          }

   [WHEN condition-2  { imperative-statement-3 } ]...
                      { NEXT SENTENCE          }
```

FORMAT 2.

```
SEARCH ALL  identifier-1  [AT END  imperative-statement-1]

WHEN { data-name-1      } IS EQUAL TO { identifier-3   } }
     { condition-name-1 }              { literal-1      }
                                       { arithmetic-expression-1 } }

[AND { data-name-2      } IS EQUAL TO { identifier-4   } }
     {                  }              { literal-2      }
     { condition-name-2 }              { arithmetic-expression-2 } ] ...

{ imperative-statement-2 }
{ NEXT SENTENCE          }
```

2.5.4.5 Procedure Division Considerations for Table Handling. (Cont.)

Identifier-1 must not be subscripted or indexed. Its description must contain an OCCURS clause with the INDEXED BY option.

Identifier-1 can be a data item subordinate to a data item that contains an OCCURS clause, thus providing for a two or three dimensional table. An index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Execution of a SEARCH statement causes modification only of the setting of the index-name associated with identifier-1 (and, if present, of index-name-1 or identifier-2). Therefore, to search an entire two or three dimensional table, it is necessary to execute a SEARCH statement several times; prior to each execution, SET statements must be executed to adjust the associated index-names to their appropriate settings.

In the AT END and WHEN options, if any of the specified imperative statement(s) do not terminate with a GO TO statement, control passes to the next sentence after execution of the imperative statement.

FORMAT 1 Considerations -- Identifier-2, when specified, must be described as an index data item, or it must be a fixed-point numeric elementary item described as an integer. When an occurrence number is incremented, identifier-2 is simultaneously incremented by the same amount.

Condition-1, condition-2, etc., may be any condition, as follows:

```
relation condition
class condition
condition-name condition
sign condition
```

Upon the execution of a SEARCH statement, a serial search takes place, starting with the current index setting.

If, at the start of the SEARCH, the value of the index-name associated with identifier-1 is not greater than the highest possible occurrence number for identifier-1, the following actions take place:

1. The condition(s) in the WHEN option are evaluated in the order they are written.

2. If none of the conditions is satisfied, the index-name for identifier-1 is incremented to reference the next table element, and step 1 is repeated.

2.5.4.5 Procedure Division Considerations for Table Handling. (Cont.)

3. If, upon evaluation, one of the WHEN conditions is satisfied, the search terminates immediately, and the imperative-statement associated with that condition is executed. The index-name points to the table element that satisfied the condition.

4. If the end of the table is reached without the WHEN condition being satisfied, the search terminates as described in the next paragraph.

If at the start of the search, the value of the index-name associated with identifier-1 is greater than the highest permissible occurrence number for identifier-1, the search is terminated immediately, and if the AT END option is specified, imperative-statement-1 is executed. If this option is omitted, control passes to the next sentence.

When the VARYING index-name-1 option is specified, one of the following applies:

1. If index-name-1 is one of the indexes for identifier-1, index-name-1 is used for the search. Otherwise, the first (or only) index-name for identifier-1 is used.

2. If index-name-1 is an index for another table entry, then when the index-name for identifier-1 is incremented to represent the next occurrence of the table, index-name-1 is simultaneously incremented to represent the next occurrence of the table it indexes.

FORMAT 2 Considerations -- The first index-name assigned to identifier-1 will be used for the search.

The description of identifier-1 must contain the KEY option in its OCCURS clause.

Condition-1 must consist of one of the following:

1. A relation condition incorporating the EQUALS, EQUAL TO, or equal sign ( = ) relation. Only the subject of the relation-condition must consist solely of one of the data-names that appear in the KEY clause of identifier-1; the object of this condition may not be a data item named in a KEY phrase.

2. A condition-name condition in which the VALUE clause describing the condition-name consists of a single literal only. The conditional variable associated with the condition-name must be one of the data-names that appear in the KEY clause of identifier-1.

3. A compound condition formed from simple conditions of the types described above, with AND as the only connective.

2.5.4.5 Procedure Division Considerations for Table Handling. (Cont.)

Any data-name that appears in the KEY clause of identifier-1 may be tested in condition-1. However, all data-names in the KEY clause preceding the one to be tested must also be so tested in condition-1. No other tests may be made in condition-1.

The example below, FIGURE 2-33, shows the use of a FORMAT 1 SEARCH statement with two WHEN options.

```
77 I PICTURE S9(4) USAGE IS INDEXED.
                .
                .
                .
Ø5 A OCCURS 1Ø TIMES ASCENDING KEY IS KEY1, KEY2, KEY3, KEY4
   INDEXED BY I.
   1Ø  KEY1 PICTURE S9.
   1Ø  KEY2 PICTURE S99.
   1Ø  KEY3 PICTURE S9.
   1Ø  KEY4 PICTURE S9.
       88  BLUE VALUE 1.
                .
                .
                .
in the Procedure Division, valid WHEN phrases could be:

    WHEN KEY1 (I) = 3 AND KEY2 (I) = 1Ø AND KEY3 (I) = 5 ...

    WHEN KEY1 (I) = 3 AND KEY2 (I) = VALUE-1
        AND KEY3 (I) = 5 AND BLUE (I) ...
```

FIGURE 2-33

During execution of a FORMAT 2 SEARCH statement, a binary search takes place; the setting of index-name is varied during the search so that at no time is it less than the value that corresponds to the first element of the table, nor is it ever greater than the value that corresponds to the last element of the table. If condition-1 cannot be satisfied for any setting of the index within this permitted range, control is passed to imperative-statement-1 when the AT END option appears or to the next sentence when this clause does not appear. In either case, the final setting of the index is not predictable. If the index indicates an occurrence that allows condition-1 to be satisfied, control passes to imperative-statement-2.

2.5.4.5 Procedure Division Considerations for Table Handling. (Cont.)

The results of a SEARCH ALL operation are predictable only when the data in the table is ordered as described by the ASCENDING/DESCENDING KEY clause associated with identifier-1

• SET STATEMENT. The SET statement establishes reference points for table handling operations by setting index-names to values associated with table elements. The SET statement must be used when initializing index-name values before execution of a SEARCH statement, it may also be used to transfer values between index-names and other elementary data items.

FORMAT.

FORMAT 1.

```
SET { index-name-1      [index-name-2] ... } TO { index-name-3 }
    { identifier-1      [identifier-2] ... }    { identifier-3 }
                                                { integer-1    }
```

FORMAT 2.

```
SET  index-name-4 [index-name-5] ...  { UP BY   } { identifier-4 }
                                      { DOWN BY } { integer-2    }
```

All identifiers must name either index data items or fixed-point numeric elementary items described as integers; however, identifier-4 must not name an index data item. When an integer is used, it may be signed; however, integer-1 must be a positive integer. Index-names are related to a given table through the INDEXED BY option of the OCCURS clause; when index-names are specified in the INDEXED BY option, they are automatically defined.

All references to index-name-1, identifier-1, and index-name-4 apply equally to index-name-2, identifier-2, and index-name-5, respectively.

FORMAT 1 Considerations -- When the SET statement is executed, one of the following actions occurs:

1. Index-name-1 is converted to a value that corresponds to the same table element to which either index-name-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place. Both before and after the execution of the SET statement the resultant value of index-name must correspond to an occurrence number of an element in the associated table.

2.5.4.5 Procedure Division Considerations for Table Handling. (Cont.)

2. If identifier-1 is an index data item, it is set equal to either the contents of index-name-3 or identifier-3, where identifier-3 is also an index data item. Integer-1 cannot be used in this case.

3. If identifier-1 is not an index data item, it is set to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor integer-1 can be used in this case.

FORMAT 2 Considerations -- When the SET statement is executed, the contents of index-name-4 (and index-name-5, etc., if present) are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of integer-2 or identifier-4.

Data in the following chart represents the validity of various operand combinations in the SET statement.

| SENDING ITEM | RECEIVING TIME | | |
|---|---|---|---|
| | Integer Data Item | Index-name | Index Data Item |
| Integer Literal | No | VALID | No |
| Integer Data Item | No | VALID | No |
| Index-Name | VALID | VALID | VALID* |
| Index Data Item | No | VALID* | VALID* |

*NOTE: No conversion takes place.

- SAMPLE TABLE HANDLING PROGRAM. Refer to FIGURE 2-34.

The program below illustrates the Table Handling feature, including the use of indexing, of the SET statement, and of the SEARCH statement (including the VARYING option and the SEARCH ALL format).

The census bureau uses the program to compare:

1. The number of births and deaths that occurred in any one of the 50 states in any one of the past 20 years with -

2. The total number of births and deaths that occurred in the same state over the entire 20-year period.

The input file, INCARDS, contains the specific information upon which the search of the table is to be conducted. INCARDS is formatted as follows:

2.5.4.5 Procedure Division Considerations for Table Handling. (Cont.)

| | |
|---|---|
| STATE-NAME | a 4-character alphabetic abbreviation of the state name |
| SEXCODE | 1 = male; 2 = female |
| YEARCODE | a 4-digit field in the range 195Ø through 1969 |

A typical run might determine the number of females born in New York in 1953 as compared with the total number of females born in New York in the past 2Ø years.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  TABLES.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-36Ø
OBJECT-COMPUTER.  IBM-36Ø.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INFILE ASSIGN TO UT-S-INTAPE.
    SELECT OUTFILE ASSIGN TO UT-S-PRTOUT.
    SELECT INCARDS ASSIGN TO UT-S-ICARDS.
DATA DIVISION.
FILE SECTION.
FD  INFILE LABEL RECORDS ARE OMITTED.
Ø1  TABLE PIC X(28200).
Ø1  TABLE-2 PIC X(1800).
FD  OUTFILE LABEL RECORDS ARE OMITTED.
Ø1  PRTLINE PIC X(133).
FD  INCARDS LABEL RECORDS ARE OMITTED.
Ø1  CARDS.
    Ø2  STATE-NAME  PIC X(4).
    Ø2  SEXCODE  PIC 9(4).
    Ø2  YEARCODE  PIC 9(4).
    Ø2  FILLER  PIC X(71).
WORKING-STORAGE SECTION.
Ø1  PRTAREA-2Ø.
    Ø2  FILLER PIC X VALUE SPACES.
    Ø2  YEARS-2Ø  PIC 9(4)
    Ø2  FILLER PIC X(3) VALUE SPACES.
    Ø2  BIRTH-2Ø  PIC 9(7).
    Ø2  FILLER PIC X(3) VALUE SPACES.
    Ø2  DEATHS-2Ø  PIC 9(7).
    Ø2  FILLER PIC X(1Ø8)  VALUE SPACES.
Ø1  PRTAREA.
    Ø2  FILLER  PIC X
    Ø2  YEAR    PIC 9(4).
    Ø2  FILLER  PIC X(3)  VALUE SPACES.
    Ø2  BIRTHS  PIC 9(5).
    Ø2  FILLER  PIC X(3)  VALUE SPACES.
    Ø2  DEATHS  PIC 9(5).
    Ø2  FILLER PIC X(112)   VALUE SPACES.
```

FIGURE 2-34

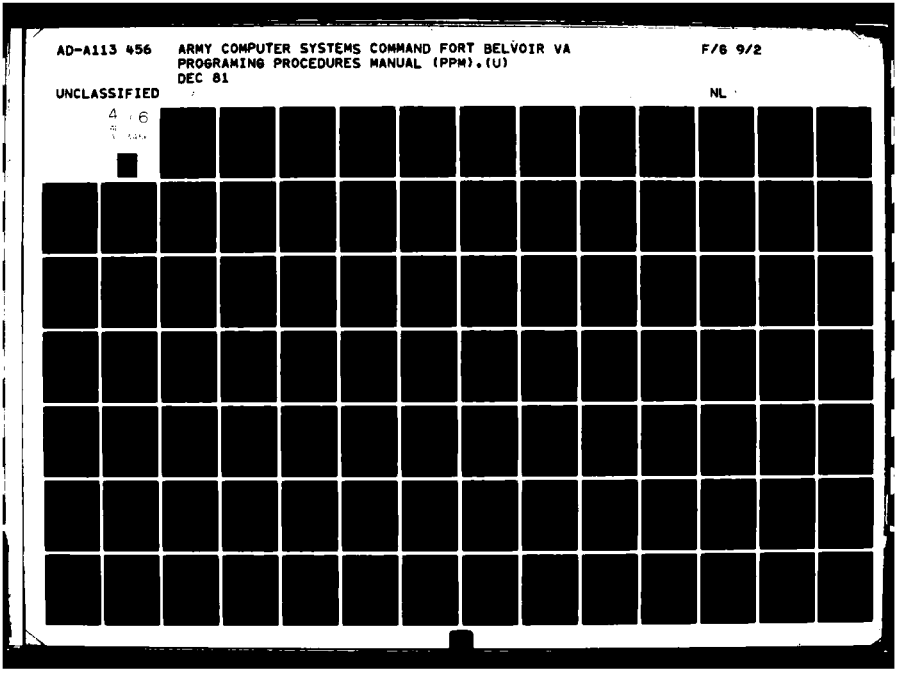
2.5.4.5 Procedure Division Considerations for Table Handling. (Cont.)

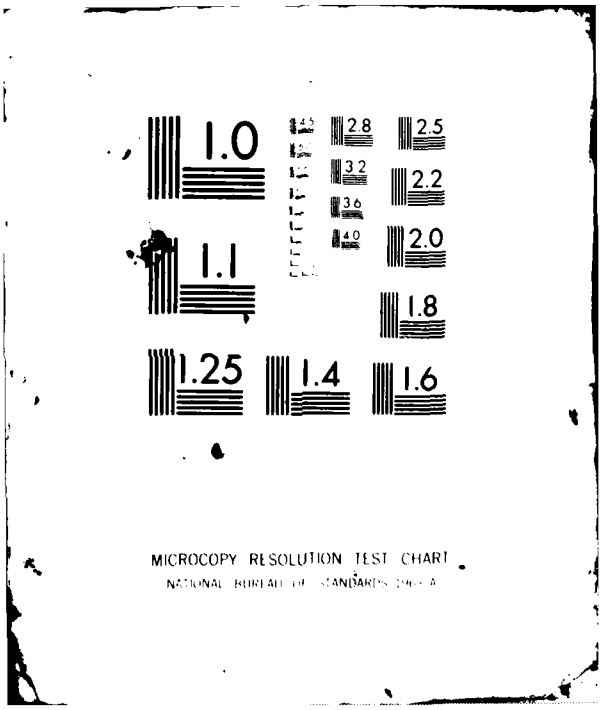
```
Ø1  CENSUS-STATISTICS-TABLE.
    Ø2  STATE-TABLE OCCURS 5Ø TIMES INDEXED BY ST.
        Ø3  STATE-ABBREV      PIC X(4).
        Ø3  SEX OCCURS 2 TIMES INDEXED BY SE.
            Ø4  STATISTICS OCCURS 2Ø TIMES ASCENDING KEY IS YEAR
                INDEXED BY YR.
                Ø5  YEAR      PIC 9(4).
                Ø5  BIRTHS    PIC 9(5).
                Ø5  DEATHS    PIC 9(5).
Ø1  STATISTICS-LAST-2Ø-YRS.
    Ø2  SEX-2Ø OCCURS 2 TIMES INDEXED BY SE-2Ø.
        Ø3  STATE-2Ø OCCURS 5Ø TIMES INDEXED BY ST-2Ø.
            Ø4  YEARS-2Ø      PIC 9(4).
            Ø4  BIRTHS-2Ø     PIC 9(7).
            Ø4  DEATHS-2Ø     PIC 9(7).
PROCEDURE DIVISION.
OPEN-FILES.
    OPEN INPUT INFILE INCARDS OUTPUT OUTFILE.
READ-TABLE.
    READ INFILE INTO CENSUS-STATISTICS-TABLE
        AT END GO TO READ-CARDS.
    READ INFILE INTO STATISTICS-LAST-2Ø-YRS
        AT END GO TO READ-CARDS.
READ-CARDS.
    READ INCARDS
        AT END GO TO EOJ.
DETERMINE-ST.
    SET ST ST-2Ø TO 1.
    SEARCH STATE-TABLE VARYING ST-2Ø AT END GO TO ERROR-MSG-1
        WHEN STATE-NAME = STATE-ABBREV (ST) NEXT SENTENCE.
DETERMINE-SE.
    SET SE SE-2Ø TO SEXCODE.
DETERMINE-YR.
    SEARCH ALL STATISTICS AT END GO TO ERROR-MSG-2
        WHEN YEAR OF STATISTICS (ST, SE, YR) = YEARCODE
            GO TO WRITE-RECORD.
```

FIGURE 2-34 (Cont.)

AD-A113 456 ARMY COMPUTER SYSTEMS COMMAND FORT BELVOIR VA F/G 9/2
PROGRAMING PROCEDURES MANUAL (PPM).(U)
DEC 81

UNCLASSIFIED NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

2.5.4.5 Procedure Division Considerations for Table Handling. (Cont.)

```
ERROR-MSG-1.
   DISPLAY "INCORRECT STATE" STATE-NAME.
   GO TO WRITE-RECORD.
ERROR-MSG-2.
   DISPLAY "INCORRECT YEAR" YEARCODE.
   GO TO READ-CARDS.
WRITE-RECORD.
   MOVE CORRESPONDING STATISTICS (ST, SE, YR) TO PRTAREA.
   WRITE PRTLINE FROM PRTAREA AFTER ADVANCING 3.
   MOVE CORRESPONDING STATE-2Ø (SE-2Ø, ST-2Ø) TO PRTAREA-2Ø.
   WRITE PRTLINE FROM PRTAREA-2Ø AFTER ADVANCING 1.
   GO TO READ-CARDS.
EOJ.
   CLOSE INFILE INCARDS OUTFILE.
   STOP RUN.
```

FIGURE 2-34 (Cont.)

USACSC GUIDELINES.

- In order to facilitate ease of program debugging and maintenance, use of compound conditional statements in conjunction with the SEARCH verb should be avoided if at all possible.

- Indexing is a more efficient means of searching a table and is the preferred procedure.

- Programers should ensure program logic includes checks to verify that subscripts and indices do not exceed maximum values as defined in the appropriate OCCURS clause.

- Subscripting will be limited to a maximum of three levels. Subscripting beyond one level should be avoided if possible.

- SEARCH ALL verb should be used instead of the SEARCH verb for large tables unless the following situations exist:

  The keys of the table being searched are not sequenced. The SEARCH ALL verb requires sequenced keys - SEARCH verb does not.

2.5.4.5 Procedure Division Considerations for Table Handling. (Cont.)

The elements of the table will have a high frequency of hits, which will take place in the same sequence as the table keys; i.e., the search can continue where the last one left off rather than needing to start at the front, and probably will require very few iterations before the next hit is made. Random searching of any large table almost always favors SEARCH ALL.

When neither of the above conditions exist, the following criteria can be used to decide when to use one version or the other. If the table contains more than 25 to 30 valid codes to be searched, the SEARCH ALL verb is more efficient. If less, use the SEARCH verb.

- Refer to USACSCM Executive Software Manual 18-2 for OS and DOS table handling routines.

2.5.5 SOURCE PROGRAM LIBRARY FACILITY.

2.5.5.1 Introduction to Copy Library Facility.

- The COBOL Source Program Library Facility provides a capability for specifying text that is to be copied from a library. These texts are available for copying at compile time using the COPY statement. The effect of the COPY statement is to insert text into the source program where it will be treated as part of the source program.

LIBRARY. The COBOL text is placed on a user maintained library. The routines for placing this text on the library and updating the text are designated by each vendor. Retrieval of the text is accomplished using the COPY statement. See paragraph 2.5.5.2.

LIBRARY MAINTENANCE AND CONTROL. Development, maintenance and control of the system/subsystem copy libraries and its members is the responsibility of the system coordinator appointed by the ASD. Entries to the library will conform to the definitions contained in the USACSC Data Element Dictionary. Once the standards are established and loaded into user libraries (copy, source, etc., library), they are easily retrieved and the standard is perpetuated throughout all programs in the system.

2.5.5.2 COPY STATEMENT.

- The COPY statement may be used in the ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. Prewritten source program entries, such as standard file descriptions, record descriptions, or procedures, may be included in the source listing without RECODING them.

- The source listing produced using the copy statement varies from vendor to vendor. In some instances, the COPY statement itself is included in the printed listing as comments with the copied text following it. In other cases, the COPY statement is completely replaced by the copied text. (See VENDORS' GUIDELINES.)

- The full format for the COPY statement is shown under Language Elements 'COPY Statement'. The general formats for use of the COPY statement are shown at FIGURE 2-35 below.

```
Option 1   (within the CONFIGURATION Section):
       SOURCE-COMPUTER.  COPY statement.
       OBJECT-COMPUTER.  COPY statement.
```

```
Option 2   (within the INPUT-OUTPUT Section):
       FILE-CONTROL.  COPY statement.
       I-O-CONTROL.  COPY statement.
```

```
Option 3   (within the FILE Section):
       FD  file-name  COPY statement.
       SD  sort-file-name  COPY statement.
```

```
Option 4   (within the DATA DIVISION):
       Ø1  data-name  COPY st
```

```
Option 5   (within the WORKING-STORAGE Section or LINKAGE Section):
       77  data-name  COPY statement.
```

FIGURE 2-35

2.5.5.2 COPY STATEMENT. (Cont.)

```
Option 6   (within the WORKING-STORAGE Section or LINKAGE Section):
      77  data-name-1  REDEFINES data-name-2  COPY statement.
      01  data-name-1  REDEFINES data-name-2  COPY statement.
```

```
Option 7   (within PROCEDURE DIVISION):
      section-name  SECTION.  COPY statement.
      paragraph-name.  COPY Statement.
```

FIGURE 2-35 (Cont.)

GENERAL RULES.

• The text is placed on the library using a unique entry name. This name is stored as a header record for identification of the entry and is not part of the text. When the text is retrieved from the library, only the text is copied, not the entry name.

• When the COPY statement is used for 01-level record descriptions or 77-level data item descriptions in the DATA DIVISION, the programer coded 01-level or 77-level name replaces the 01-level or 77-level name from the library on the compiled source listing.

If a reference is made in an OCCURS ... DEPENDING ON clause to the original 01 or 77-level name on the library, this reference must be changed by using the REPLACING option of the COPY statement. (See EXAMPLES 2 and 3 of FIGURE 2-36.)

The library name and text are unchanged in the library entry.

• When the COPY statement is used in the PROCEDURE DIVISION, the copy statement as written by the programer is completely replaced by the copied text. (See EXAMPLE 4 of FIGURE 2-36.)

VENDORS' GUIDELINES.

IBM.

• When the COPY statement is used in the ENVIRONMENT DIVISION or for catalogued file descriptions in the DATA DIVISION, the compiler prints out the COPY statement as written by the programer with the copied text following it. (See EXAMPLE 1 of FIGURE 2-36.)

2.5.5.2 COPY STATEMENT. (Cont.)

• IBM allows the retrieval of the data from the copy library two ways: the use of the COPY statement in the source coding and the use of the BASIS card in the Job Control Language (JCL) in the compile deck.

• The IBM compiler flags each statement copied from a library with a 'C' preceding the sequence number on the resultant source listing.

EXAMPLES. The following examples in FIGURE 2-36 illustrate the use of the COPY statement. The examples start in column 7. Any vendor's indicators flagging copied statements are not shown.

USACSC GUIDELINES. None.

2.5.5.2 COPY STATEMENT. (Cont.)

```
*EXAMPLE 1.
*    THIS EXAMPLE ILLUSTRATES THE COPYING OF AN FD.


*    THE FOLLOWING ENTRY IS A MEMBER OF A COPY LIBRARY.
*    THIS ENTRY HAS A LIBRARY ENTRY NAME OF PAYFILEA.

     LABEL RECORDS ARE STANDARD
     RECORDING MODE IS F
     RECORD CONTAINS 100 CHARACTERS
     BLOCK CONTAINS 500 CHARACTERS
     DATA RECORD IS PAYREC-A.
 01  PAYREC-A                        PIC X(100).


*    THE FOLLOWING CODING WAS INCLUDED IN A SOURCE LISTING.

 FD  PAYFILE COPY PAYFILEA.


*    THE FOLLOWING IS THE RESULTANT CODING PRODUCED BY THE
*    COMPILER FROM THE ABOVE LIBRARY ENTRY AND COPY STATEMENT.

 FD  PAYFILE COPY PAYFILEA.
 FD  PAYFILE
     LABEL RECORDS ARE STANDARD
     RECORDING MODE IS F
     RECORD CONTAINS 100 CHARACTERS
     BLOCK CONTAINS 500 CHARACTERS
     DATA RECORD IS PAYREC-A.
 01  PAYREC-A                        PIC X(100).
```

FIGURE 2-36

2.5.5.2 COPY STATEMENT. (Cont.)

```
*EXAMPLE 2.
*    THIS EXAMPLE ILLUSTRATES THE COPYING OF Ø1-LEVEL RECORD
*    DESCRIPTIONS.


*    THE FOLLOWING ENTRY IS A MEMBER OF A COPY LIBRARY.
*    THIS ENTRY HAS A LIBRARY ENTRY NAME OF PAYREC-2.

 Ø1  PAYREC-2.
     Ø5  PAYREC-GRADE            PIC 99.
     Ø5  PAYREC-RATE             PIC S9(5) COMP-3.
     Ø5  PAYREC-HOURS            PIC S9(3) COMP-3.
     Ø5  PAYREC-CODE           OCCURS 1 TO 18 TIMES
                                     DEPENDING ON
                                     PAYREC-GRADE OF PAYREC-2
                                 PIC XX.
     Ø5  FILLER                  PIC X(57).


*    THE FOLLOWING CODING WAS INCLUDED IN A SOURCE LISTING.

 Ø1  PAYREC-B COPY PAYREC-2.


*    THE FOLLOWING IS THE RESULTANT CODING PRODUCED BY THE
*    COMPILER FROM THE ABOVE LIBRARY ENTRY AND COPY STATEMENT.

 Ø1  PAYREC-B COPY PAYREC-2.
 Ø1  PAYREC-B.
     Ø5  PAYREC-GRADE            PIC 99.
     Ø5  PAYREC-RATE             PIC S9(5) COMP-3.
     Ø5  PAYREC-HOURS            PIC S9(3) COMP-3.
     Ø5  PAYREC-CODE           OCCURS 1 TO 18 TIMES
                                     DEPENDING ON
                                     PAYREC-GRADE OF PAYREC-2
                                 PIC XX.
     Ø5  FILLER                  PIC X(57).


*    NOTE THAT THE QUALIFIER PAYREC-2 IN THE DEPENDING ON OPTION
*    OF THE OCCURS CLAUSE DID NOT CHANGE.  THIS MUST BE ACCOMPLISHED
*    USING THE REPLACING OPTION OF THE COPY STATEMENT.
```

FIGURE 2-36 (Cont.)

2.5.5.2 COPY STATEMENT. (Cont.)

```
*EXAMPLE 3.
*    THIS EXAMPLE ILLUSTRATES THE USE OF THE REPLACING OPTION OF
*    THE COPY STATEMENT.  THE COPY LIBRARY MEMBER FROM EXAMPLE 2
*    IS USED FOR THIS EXAMPLE.

*    THE FOLLOWING CODING WAS INCLUDED IN A SOURCE LISTING.

 01  PAYREC-B COPY PAYREC-2 REPLACING PAYREC-2
                              BY PAYREC-B.


*    THE FOLLOWING IS THE RESULTANT CODING PRODUCED BY THE
*    COMPILER FROM THE ABOVE LIBRARY ENTRY AND COPY STATEMENT.

 01  PAYREC-B COPY PAYREC-2 REPLACING PAYREC-2
                              BY PAYREC-B.
 01  PAYREC-B.
     05  PAYREC-GRADE            PIC 99.
     05  PAYREC-RATE             PIC S9(5) COMP.
     05  PAYREC-HOURS            PIC S9(3) COMP.
     05  PAYREC-CODE           OCCURS 1 TO 18 TIMES
                                     DEPENDING ON
                                     PAYREC-GRADE OF PAYREC-B
                                PIC XX.
     05  FILLER                  PIC X(57).


*    NOTE THAT PAYREC-2 HAS NOW BEEN CHANGED TO PAYREC-B IN ALL
*    OCCURRENCES IN THE RECORD.
```

FIGURE 2-36 (Cont.)

2.5.5.2 COPY STATEMENT. (Cont.)

```
*EXAMPLE 4.
*    THIS EXAMPLE ILLUSTRATES THE COPYING OF PROCEDURES INTO A
*    SOURCE PROGRAM.


*    THE FOLLOWING ENTRY IS A MEMBER OF A COPY LIBRARY.
*    THIS ENTRY HAS A LIBRARY ENTRY NAME OF STATE-TAX-RT.

     MULTIPLY PAYREC-RATE BY PAYREC-HOURS
         GIVING TOTAL.
     MULTIPLY TOTAL BY .Ø15 GIVING STATE-TAX.


*    THE FOLLOWING CODING WAS INCLUDED IN A SOURCE LISTING.

 ØØ1Ø-ST-TAX-RTN.  COPY STATE-TAX-RT
     REPLACING MULTIPLY TOTAL BY .Ø15 GIVING STATE-TAX
     BY MULTIPLY TOTAL BY .Ø2Ø GIVING STATE-TAX.


*    THE FOLLOWING IS THE RESULTANT CODING PRODUCED BY THE
*    COMPILER FROM THE ABOVE LIBRARY ENTRY AND COPY STATEMENT.

     MULTIPLY PAYREC-RATE BY PAYREC-HOURS
         GIVING TOTAL.
     MULTIPLY TOTAL BY .Ø2Ø GIVING STATE-TAX.
```

FIGURE 2-36 (Cont.)

## 2.5.6 DEBUGGING AIDS.

### 2.5.6.1 Introduction to Debugging Aids.

DEBUGGING FEATURES. The debugging features available in COBOL are designed to aid the programer in producing an error-free program in the shortest possible time. Further, the debugging features are designed to permit most debugging to be accomplished at the source language level. Debugging in COBOL can occur at two points in the program production process: during compilation and during execution of the object program. In general, debugging during compilation deals primarily with the removal of syntax errors from the source program, while debugging during execution deals with the removal of logical errors from the object program. For the most part, the COBOL compiler detects syntax errors in the source program and advises the programer of these during the compilation process. The programer can then correct the source program and recompile, repeating this cycle until the compiler indicates that an error-free compilation has occurred.

At this point, the programer is ready to test and debug the object program using test data. The compiler provides a debugging language feature for this phase of the debugging. This feature enables the programer to insert source language statements that, when compiled and executed as part of the source program, will produce output showing the flow of control and the values of selected items as the program is being executed. Based on this output, the programer can correct the source program, recompile, execute, and reevaluate the new output. When an error-free object program has been obtained, the debugging statements are removed from the source program and the program is recompiled producing an object program ready for execution in a production mode.

DESK-CHECKING. The whole idea of debugging is to produce a program that is free of errors. There are other things that do aid in detecting errors and can therefore be included under the topic of debugging. One such method is simple desk-checking. The following is a list of things that should be double checked since they have consistently been causes for errors in previous programs.

- Problems that arise in many programs are the result of:

  Missing or misplaced periods.

  Files not being properly opened and closed.

  Duplicate paragraph names.

  Misspelled data-names, reserved words and procedure-names.

  Improper characters in names or more than 30 characters in a name.

2.5.6.1 Introduction to Debugging Aids. (Cont.)

Improper use of the VALUE IS clause. The VALUE IS clause:

Must be compatible with the class of the item.

Must not be used in an entry with an OCCURS clause.

Must not be used in an entry subordinate to an OCCURS clause.

Must not be used in the FILE SECTION of the DATA DIVISION.

Must not be used in an entry containing an edit picture.

Misuse of the OCCURS clause. The OCCURS must:

Not be used in entries with levels Ø1, 77, 88.

An item defined with an OCCURS clause must not be referenced without a subscript.

Alphanumeric literals requiring more than one line must have a hyphen (-) in column 7 and a quote (') in column 24 of the continuation line.

Misuse of the REDEFINES clause. In the REDEFINES clause:

Multiple redefinitions must be at the same level with no entries intervening at the same level.

Explicit redefinition may not occur at the Ø1 level in the FILE SECTION.

Values of subscripts outside of their range: The values in a subscript should be checked to assure you that the subscript has not gone outside of its range.

DEBUGGING FACILITY. The debugging facility consists of the use of the USE FOR DEBUGGING declarative or individual debugging lines. See paragraph 2.4.9.10.

DEBUGGING LINES

A debugging line is any line in a source program with a "D" coded in column 7 (the continuation area). If a debugging line contains nothing but spaces in Area A and Area B, it is considered a blank line.

2.5.6.1 Introduction to Debugging Aids. (Cont.)

Each debugging line must be written so that a syntactically correct program results whether the debugging lines are compiled into the program, or treated as documentation.

Successive debugging lines are permitted. Debugging lines may be continued; however, each continuation line must contain a "D" in column 7, and character-strings must not be broken across two lines.

Debugging lines may be specified only after the OBJECT-COMPUTER paragraph.

When the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph, all debugging lines are compiled as part of the object program.

When the DEBUGGING MODE clause is omitted, all debugging lines are treated as documentation.

2.5.6.2 DOS COBOL Program Debugging Aids.

- The following are a few pointers to aid the programer in debugging DOS COBOL programs. The necessary tools for debugging from a core dump are: program post list, core dump, and Linkage Editor Map.

- Determine the COBOL statement that generated the address of the program check.

The top of the system dump will tell you the address of the program check and the type of program check. Locate the instruction in the core dump.

Determine the relocation factor of your program from the Linkage Editor Map. Subtract the relocation factor of your program in the Linkage Editor Map from the address of the offending instruction.

The address that results may be located in the Procedure Division Map generated at compile time.

Preceding the address and code found in step three, you will find the sequence number of the corresponding COBOL statement in the post list and the number of the element in the sentence that generated the code.

- Determine if the COBOL statement is coded incorrectly.

- If the statement is coded correctly, go back to the core dump and find out the type of program check.

If it is a data exception, you will probably find that the instruction is a decimal instruction and one of the fields either will not have a valid sign or

2.5.6.2 DOS COBOL Program Debugging Aids. (Cont.)

will contain digits other than Ø through 9. To determine this, it will be necessary to find the fields in core.

Inspect bits 4-7 of the low order byte for a valid sign. If one is not present, this is the cause of the program check.

2.5.6.3 Common Causes of Errors.

- Blanks in fields defined as numeric.

- Failing to initialize counters and thus not insuring a valid beginning value.

- Moving zeros to a group level item to zero several fields -- a valid sign is generated only for the lowest order field.

- Adding into a field that is subscripted can cause trouble if care is not taken to insure that the subscript does not get too large.

If it is a protection exception, one possible cause is that a base register used in the instruction has not been initialized. Base registers in COBOL are initialized at different times. For input files, the register is not initialized until the first successful read; they are not initialized when the files are opened. For output files, the registers are initialized when they are opened. When faced with a protection exception, go to the COBOL post list and check to be sure that no data has been moved prior to the time when base registers will be initialized.

If an addressing or specification exception occurs, you may (but not always) find upon inspection that registers have been unexpectedly modified and the problem becomes one of the finding out how. A possible approach is: Check the addresses in register fourteen and fifteen against the address of the program check instruction. If the address of the program check instruction is equal or slightly larger than the address in register fifteen, you are probably in a subroutine and the address in register fourteen should be your return address. A BAL or BALR instruction will probably precede your return address. Look for this particularly when the problem does not seem to be with a COBOL statement.

- If the job is canceled because of an illegal supervisor call (SVC 32), two of the possible causes are a wrong length record condition occurred or the problem program says it will handle a file in a certain manner when, in fact, it does not. One method of investigating the cause of this type of problem is:

Determine the module that contains the SVC 32 instruction from the error address given with the error message (first page DOS core dump) and the relocation factors on the Linkage Editor Map.

2.5.6.3 Common Causes of Errors. (Cont.)

If the instruction is located in the lower problem program area, you probably have a wrong length record error. The file name (SYS number) of the file on which this occurred will be located 98 bytes (Hex '62') ahead of the SVC 32.

If the instruction is located in a logic module, obtain the listing of the module from the error address. The resulting number is the address of the error instruction in the module post list. After locating the offending instruction, you will often find comments which will point you toward the source of the trouble.

Failure to clear the output area, clearing the output area before the first record is written only clears the first record area within the output block.

Failure to insure that a subscript does not exceed the range of the associated OCCURS clause may lead to referencing incorrect storage locations.

Branching out of a perform and not exiting properly may cause trouble.

Moving any data to or from any I/O area before opening the file, and in the case of input files before issuing the first successful read.

Attempting to read from a file which has already taken the 'AT END' branch will cause trouble for the programer.

2.5.6.4 Link Edit Map.

- The PHASE, on a Link Edit Map, is the name by which a program was cataloged to the Core Image Library. It is the name by which a program phase is retrieved from the Core Image Library, brought into main storage and executed, when the name is the phase-name parameter of an EXEC control statement. Programs that are link-edited and executed immediately, from the transient area of the Core Image Library, are given the name PHASE***, as shown on the example link edit map. This is merely a dummy name for all phases that are tested before cataloging to a permanent location on the Core Image Library.

- XFR-AD is the abbreviation for TRANSFER ADDRESS. This column on the link edit map will show the absolute address of the first instruction of the COBOL program's Procedure Division in main storage when the program is to be executed.

- LOCORE indicates the absolute address, in main storage, of the beginning of the area immediately following the LABEL PROCESSING area.

- HICORE indicates the absolute address of the last instruction of the link-edited program.

- DSK-AD is the disk address, on the systems residence disk pack of the program.

2.5.6.4 Link Edit Map. (Cont.)

• ESD TYPE - External Symbol Dictionary. This column designates the elements, or Control Sections that make up a program -- with sub-elements or entries, whether program phases or overlays.

• LABEL - An eight character name of a control section, entry, overlay or element that is part of a program phase.

• Each character position, or group of character positions, of an IBM module has specific meaning. The first two positions identify the type of module; the third position contains a code which represents the media of the file; the fourth position will contain a code, for data file modules, which represents the recording mode of the records in the file; the fifth through eighth positions indicate options such as:

Control character.

Input or output.

How many I/O areas.

Device.

Error options.

Checkpoint options.

• LOADED shows the absolute address of the module or element in the executable program.

• REL-FR is the abbreviation for relocation factor.

2.5.6.5 Object Storage Layout.

• The relative positions of the components of a COBOL program in core will generally be in the following sequence within the background partition. See FIGURE 2-37 below.

2.5.6.5 Object Storage Layout. (Cont.)

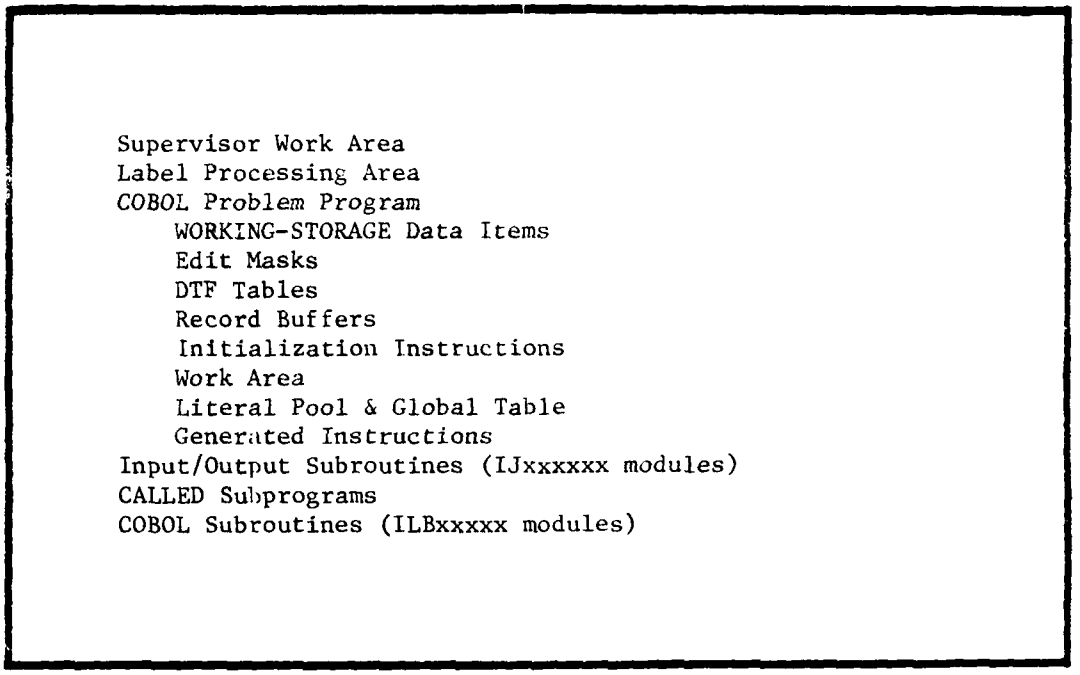



FIGURE 2-37

SUPERVISOR WORK AREA.

• This area, a part of the supervisor, is for special groups of instructions, or subroutines, which the supervisor calls in from the Core Image Library as needed for special functions. After the instructions are executed, other subroutines are called in again, as needed.

• One of the subroutines most frequently called into this area is the 'Attention Routine'. Brought in by depressing the Request Key on the Console Typewriter, this is the communication media between the supervisor and the console operator. Other subroutines called in when needed are: outputting a core dump and opening and closing files.

LABEL PROCESSING AREA.

• While a program is executing, file labels have to be either checked to ensure that the proper files are being accessed, or created if the files do not yet exist.

2.5.6.5 Object Storage Layout. (Cont.)

• This checking or creating is accomplished by the special Logical Input-Output Control System (LIOCS) module that manipulates the particular file. These are special LIOCS modules for each type of file: tape, sequential disk, card, print, index sequential, etc.

Most of these special modules have their own working-storage area in which labels can be stored while checking or creating them. The two exceptions are the module that manipulates a tape file and the module that manipulates an index sequential file. Therefore, when a program has either or both of these types of files, a special area has to be set aside at Link Edit time called the 'Label Processing Area'. This is done by inserting a // LBLTYP card in the job stream immediately preceding the // EXEC LNKEDT card. This will allow the Linkage Editor program to reserve the proper amount of additional core storage needed for processing the labels of the tape or index sequential files. The // LBLTYP card must contain a parameter that describes the maximum amount of additional core storage needed for this area. If the processing program has a tape file, the parameter NSD(nn), where nn is the largest number of extent for a single file, should be included in the // LBLTYP card. This will reserve 80 additional bytes of core storage for label processing. If the processing program has an index sequential file, 84 bytes will be reserved plus 20 additional bytes for each // EXTENT card. If the processing program has both tape and index sequential files, only one // LBLTYP card is needed -- with the parameter describing the largest file -- the index sequential file.

COBOL SUBROUTINES.

• The Linkage Editor program has the capability of recognizing that a program phase needs subroutines that were not recognized as needed by the compiler. If the compiler had seen the need for these routines, it would have created an INCLUDE card to alert the Linkage Editor of the need and caused the subroutine to be pulled from the relocatable library at Link Edit time. One type of these modules is the ILB series of relocatable modules.

WORKING-STORAGE.

• The area designated 'Working-Storage' is the beginning of the COBOL problem program. This beginning point will be related to a location in the Link Edit Map, which will be discussed later.

• The working-storage area contains those constants, accumulators, headers and other data defined in the WORKING-STORAGE SECTION of the COBOL problem program.

EDIT MASKS.

• Edit Masks will be present whenever the program has any items described with a report usage, such as floating dollar signs, leading zero suppression, check protection, credit or debit signs and others.

2.5.6.5 Object Storage Layout. (Cont.)

• Edit Masks are constructed by the compiler and are used whenever a numeric item is moved to the report item for which the edit mask was created. The edit mask is moved to a work area before the numeric move so the mask will not be destroyed and can be used again when needed.

DTF TABLES.

• DTF is the abbreviation for Define the File. There will be a DTF table for every logical file in a program. The table will contain much meaningful information concerning the file.

Some of this information is:

The symbolic name of the file.

The address of the LIOCS module that manipulates the file.

The type of device on which the file resides.

The access method.

The organization of the file.

The record length.

The block length.

The key location and length -- if index sequential.

The current record address.

The current block address.

The alternate block address.

Some of this information is inserted at compile time, other information at execution time. Naturally, information in the DTF changes as records and blocks are accessed.

ANSI COBOL, DECIPHERING DTFs.

• GENERAL. Conveys general information about DTFs and covers the following topics: definition of DTFs, core storage layout, general location of DTFs, exact address of DTFs and contents of DTFs.

• DEFINE THE FILE (DTF). In a COBOL program each file is described by statements in the ENVIRONMENT and DATA DIVISIONS. The COBOL compiler analyzes the file description entries and produces an area of "summary information" for

2.5.6.5 Object Storage Layout. (Cont.)

each file described. The file descriptors define the file, so the "summary information" is called the DTF for the file. The area will vary depending on the device-class (unit-record, utility, direct-access); access method (input, output, I/O); file organization (direct, sequential, indexed); record/block length; and presence or absence of an alternate record/block area.

• GENERAL LOCATION OF DTFs. DTFs are located beginning a few bytes beyond the last element of the area generated by the WORKING-STORAGE SECTION of a COBOL program--always on a double word boundary (last hex digit of the core location will be either Ø or 8). This, one method of locating the general area of the DTFs is to add the length of the WORKING-STORAGE SECTION (from the DATA DIVISION MAP) to the load point for the COBOL program (from the linkage editor map). The sum obtained will usually be within 5Ø bytes of the first DTF for the program in core. The DTFs will be in the same order as the FD entries in the COBOL source program.

• EXACT ADDRESS OF DTFs. To locate the exact address of a particular DTF, the following procedure may be used if a current Source Statement Listing, Procedure Division Map, and core dump are available.

In the Source Statement Listing, locate a READ, WRITE, or REWRITE statement for the file in question.

Using the line number for the READ, WRITE or REWRITE statement in the Source Statement Listing, find the generated machine-language instructions for that line in the Procedure Division Map.

In the generated machine language instructions, find the first machine language instruction beginning with '41' (Load Address) for COBOL verb READ or '58' (Load) for COBOL verb WRITE. The corresponding DTF is identified here in each case under the column of compiler generated information about the operands of the generated instructions.

Add the value in the base register (from the first page of the core dump) to the displacement value in the '41' Load Address or '58' Load instruction.

The sum of the two values is the hexadecimal location of the beginning of the DTF in the core dump.

NOTE: The above method will work if the register settings have not been changed by subroutines, IOCS modules, or subprograms.

• CONTENTS OF THE DTF. Once located, the DTF reveals information which can be quite useful in debugging a program having I/O problems. Indicators in the DTF reveal the status of the file at the time the core dump occurred. In the

2.5.6.5 Object Storage Layout. (Cont.)

following discussion, consideration is given to only those areas of the DTF which might help to determine the cause of a file handling problem. (Refer to FIGURE 2-38.)

NOTE: Both byte locations and contents are expressed in hexadecimal notation relative to zero.

| BYTES | CONTENTS |
|---|---|
| Ø-1 | The first two bytes of DTF are used as a counter. For an input file, the number of bytes transferred from the input device is counted into bytes Ø-1. After the block of records has been transferred, the count in bytes Ø-1 is subtracted from the block size specified for the file. Thus, if these bytes have a non-zero count, the block transferred was shorter than what was specified in the RECORD and BLOCK CONTAINS clauses of the FD entry for the file. When a non-zero situation occurs in these two bytes, the system takes one of two actions. If the non-zero count is an even multiple of the record size (such as the short block written at the end of a tape or sequential disk file), file processing will continue in a normal manner, taking the AT END instruction after the last record in the short block has been read. A non-zero size causes automatic and immediate program termination with a core dump. The reason given for the dump will appear on both the console and the printer as ILLEGAL SUPERVISOR CALL 32. Generally, the hex core location of the 'ØA 32' instruction will be within the DTF having a non-zero count. Note that when a block of records longer that what was described in the FD is encountered while processing a file, these bytes will contain 'ØØØØ' but a core dump termination will still occur since no core is available for the additional data waiting to be transferred. |
| 6-7 | For COBOL files, byte 6 will contain 'Ø1' indicating an open file. Byte 7 will hold the 'SYS' number assigned to the file in the ENVIRONMENT DIVISION for unit-record and tape files. For disk files, the SYS number in the EXTENT card for the prime data area will appear in byte 7. |

FIGURE 2-38

2.5.6.5 Object Storage Layout. (Cont.)

| BYTES | CONTENTS |
|---|---|
| 11-13 | These bytes point to the address of the logical IOCS module being used by the file. The address should match the load point of one of the IJxxxxxx subroutines listed in the LINKAGE EDITOR MAP. |
| 14 | Byte 14 is the DTF "type" byte. It identifies the device-type, access method, and organization of the file. FIGURE 2-39 shows the more common configurations for this byte. |
| 16-1B | For disk and tape files only, these bytes will contain the letters "SYS" followed by the assigned numbers for the file from the SELECT entry. |

FIGURE 2-38 (Cont.)

• The rest of the DTF is highlighted in FIGURE 2-40 since, beyond what has been mentioned above, the address locations for file elements are in different positions for different types of files. Add the value found in FIGURE 2-40 to the location of the first byte of the DTF to determine the data element sought.

EXAMPLE: To find the address of the current record in a sequential disk file, add hex '58' to the beginning location of the DTF. At DTF + '58' will be found the address of the first byte of the record being processed at the time the core dump occurred. For additional information, see the Sample Problem.

2.5.6.5 Object Storage Layout. (Cont.)

| HEX VALUE | FILE TYPE |
|---|---|
| Ø2 | Card Reader |
| Ø3 | Console |
| Ø4 | Card Punch |
| Ø8 | Printer |
| 1Ø | Unlabeled Tape |
| 11 | Non-Standard Labeled Tape |
| 12 | Labeled Output Tape |
| 14 | Labeled Input Tape |
| 2Ø | Sequential Disk |
| 22 | Direct-Access (LOADA) |
| 23 | Device Independent File |
| 24 | Index-Sequential (LODIS) |
| 26-27 | Index-Sequential (SRUIS, RRUIS, RUAIS, RSAIS) |
| 34 | Direct Access File |

CONFIGURATIONS FOR BYTE X'14' FOR OPEN FILES

FIGURE 2-39

2.5.6.5 Object Storage Layout. (Cont.)

| | | |

2.5.6.5 Object Storage Layout. (Cont.)

GENERAL NOTES:

1. Some of the values shown above will not be present until after the file is opened.

2. All values above are for fixed-length records only.

3. B = 1 byte, H = 2 bytes, F = 4 bytes.

SPECIFIC NOTES.

1. The block length shown in the DTF will be the actual block length + 8.

2. The data begins 8 bytes beyond the address given in the DTF.

3. Key location figure is valid only when records are blocked.

4. The address will be valid only when bytes '64-65' contain 'ØØDC' and the program has specified sequential retrieval.

RECORD BUFFERS.

- These are the areas into which input data is stored after retrieval from the physical file or from which output data is retrieved when being placed on a physical file.

- It is normal to have two areas for each file justifying the term 'double buffered'. This allows one area to be filled with information or data while the other area is being accessed or, for output files, one area to be outputed while the other is being filled.

- Programs which are very large and are considered 'core-bound', or so large as to need almost all of the available core storage, can limit buffers to one, but this reduces efficiency as the one buffer cannot be refilled until it has been completely exhausted.

- Input/output buffers should be as large as a program will permit so that a maximum number of characters can be transmitted between the device and core storage. This produces the greatest efficiency.

WORK AREA, TASK GLOBAL TABLE (TGT), PROGRAM GLOBAL TABLE (PGT), LITERAL POOL.

- The work area is used during execution of a COBOL program for arithmetic calculations having intermediate results, for editing numeric fields being moved to a report receiving field -- using one of the edit masks -- and by some of the more complex COBOL verbs such as GO TO ... DEPENDING ON.

2.5.6.5 Object Storage Layout. (Cont.)

• The literal pool is generated by the compiler whenever literal values, rather than data-names are used in the PROCEDURE DIVISION of a COBOL program. There are two types of literals: numeric -- utilizing the numbers Ø through 9 and non-numeric -- utilizing any of our 256 character set inclosed in quote marks. Numeric literals have a size limit of 18 digits while non-numeric literals may be 12Ø characters in length.

• The global tables are used as directories to show core positions or absolute addresses assigned to program elements or modules by the Linkage Editor program. At compile time, the compiler only sets up linkage to these elements, as their absolute address cannot be known until Link Edit time.

COMPILED INSTRUCTIONS - PROBLEM PROGRAM.

• This area contains the object language instructions generated by the compiler from the COBOL source statements of the problem program.

• Machine language generated by the compiler will appear in the core dump.

LIOCS MODULES.

• LIOCS is the abbreviation for Logical Input Output Control System. Some of the modules that make up this group of elements are the file manipulating modules. These are the instructions that LIOCS will execute when reading, writing or rewriting a file. There are special modules for each of the following file types:

Card Reader.

Print File.

Card Punch.

Tape.

Sequential Disk Input.

Sequential Disk Output.

Index sequential being created (output, ISC).

Indexed sequential extend (input, ISE).

• When a program has more than one tape file or more than one disk file, the same module will probably be used to manipulate each file, but each will have its own DTF table as previously discussed.

2.5.6.5 Object Storage Layout. (Cont.)

CALLED SUBROUTINES. The Procedure Division of a COBOL program can 'CALL', from the Relocatable Library, subroutines that exist there which will perform a special function. The Compiler, while compiling the source statements of a COBOL program, will create INCLUDE cards, which will alert the Linkage Editor program to pull these subroutines from the Relocatable Library and make them part of a program phase. There are many preconstructed subroutines on the Relocatable Library that can be used by a program. Two examples are: P51ATP, which retrieves today's date from the communication area of the supervisor, and P61ATP, which stores records on a tape which may be later outputed to the printer.

2.5.6.6 System Action Under Cancel.

- The following lists all cancel codes and their message prefixes. Some do not appear in a foreground PIB, such as the HEX 'FF' code (supervisor catalog failure). This type of function can be performed only in the background partition. The linkage editor and system maintenance functions must also be performed in the background area.

- Byte 1 of the PIB table contains a cancel code stored by the system any time a cancel condition is encountered. The PIB table can be located by displaying on the console the communication region address (located at HEX 16-17) plus the displacement of a '5A' and '5B'. This is the address of the first part of the PIB table. Remember each entry is 16 decimal (HEX '1Ø') bytes in length. Each type of the PIB is numbered starting with Ø and continuing through 15.

- Cancel Code (Hexadecimal): 1Ø

| Message Code: None |
|---|

Description, Action or Condition: This is normal end of job (EOJ). Cancel code X'1Ø' is posted in byte 1 of the PIB for the program issuing the SVC 14. The next time the canceled program is selected on general exit, an SVC 2 is taken to call in a B-transient program, which, in turn, calls job control to perform the end-of-job step.

- Cancel Code (Hexadecimal): 17

| Message Code: OSO21 |
|---|

2.5.6.6 System Action Under Cancel. (Cont.)

Description, Action or Condition: This is caused by the main task in a partition issuing the CANCEL macro without detaching all subtasks running under its control.

- Cancel Code (Hexadecimal): 18

| Message Code: None |
|---|

Description, Action or Condition: This is caused by the main task issuing the DUMP macro with subtasks attached. It allows the dump to take place without the error cancel message being printed. All subtasks are detached and EOJ is taken after the dump is complete.

- Cancel Code (Hexadecimal): 19

| Message Code: OP741 |
|---|

Description, Action or Condition: This is caused by the operator responding to an I/O error message with the cancel option on the 1Ø52.

- Cancel Code (Hexadecimal): 1A

| Message Code: OP731 |
|---|

Description, Action or Condition: This is caused by an I/O error that cannot be handled by the program (task), thus causing the program to be canceled. If the DUMP option is specified at system generation time, a dump of the supervisor and the partition in which the program was running will be taken.

- Cancel Code (Hexadecimal): 1B

| Message Code: OP821 |
|---|

2.5.6.6 System Action Under Cancel. (Cont.)

Description, Action or Condition: This is caused by a channel failure.

- Cancel Code (Hexadecimal): 1C

| Message Code: OS141 |
|---|

Description, Action or Condition: This is caused by a subtask issuing the CANCEL ALL macro. This causes all other subtasks to be detached and canceled. The main task is canceled, and a dump of the supervisor and partition involved results.

- Cancel Code (Hexadecimal): 1D

| Message Code: OS121 |
|---|

Description, Action or Condition: This is caused when the main task terminates before all subtasks have been detached. This indicates the subtasks were canceled before they came to a normal EOJ. The subtasks are detached, and the complete partition is canceled.

- Cancel Code (Hexadecimal): 1E

| Message Code: OS131 |
|---|

Description, Action or Condition: This is caused by the combination of one task issuing an enqueue for a resource, and another task issuing a dequeue for that same resource. As a result, the previous owner cannot be identified because register Ø in the save area has been modified.

- Cancel Code (Hexadecimal): 1F

| Message Code: OP81I |
|---|

2.5.6.6 System Action Under Cancel. (Cont.)

Description, Action or Condition: This is caused by a CPU failure.

- Cancel Code (Hexadecimal): 2Ø

| Message Code: OS031 or OS11I |
|---|

Description, Action or Condition: This is caused by a program check interrupt. The program is canceled by the system. The user may supply a PC or AB routine to handle this condition via the STXIT macro. This code is also used when a routine in the transient area is canceled due to a program check in the task or subtask using it.

- Cancel Code (Hexadecimal): 21

| Message Code: OS04I or OS09I |
|---|

Description, Action or Condition: This can be caused by many user errors.

- The complete text for message OS041 is:

| ILLEGAL SVC - HEX LOCATION nnnnnn - SVC CODE nn |
|---|

where nn is in hexadecimal notation. This message results from the following causes:

When nn is Ø2: The phase name given does not start with $$B, or

For LIOCS, macros called in invalid sequence. As a result, an SVC 8 is issued after an SVC 2 before an SVC 9 has been issued to free the transient area, or for other conditions, the user specified a temporary exit (SVC 8) for a logical transient. In the temporary exit routine, another routine is called (by an SVC 2) before an SVC 9 is issued to free the transient area.

When nn is Ø5: The 'to' range specified in the MVCOM macro is invalid, or MVCOM macro was issued by a foreground program, operating under single program initiation.

2.5.6.6 System Action Under Cancel. (Cont.)

When nn is 0A, 12, 13, or 18: The interval timer was not allocated to this partition, or the supervisor was generated without the timer option.

When nn is 0B: The call was not given by a logical transient routine.

When nn is 16, 17, or 1A: The caller did not have a PSW key of zero. This is applicable only in a multiprograming system.

When nn is 23: More than 16 holds have been issued for the same track.

When nn is 24: Free a non-DASD or a track that is not held.

When nn is 26: A subtask issued attach, or the save area is not on a doubleword boundary.

When nn is 27: A main task issued detach without SAVE = parameter, or a main task issued detach, but the ID of the subtask in the save area passed is not valid, or if a main task attempts to detach an already terminating subtask.

When nn is 29: A DEQ is issued by a task that did not ENQ the resource. (This is valid in an AB routine.)

When nn is 2A: A subtask (without an ECB = parameter) has issued an ENQ macro, or a subtask has issued an ENQ macro to a resource that has been dequeued by another task that has been terminated, or a task has issued two ENQ macros to the same resource without an intervening DEQ.

When nn is 2D: Emulator execution was attempted, but the EU parameter of the SUPVR macro was omitted or incorrectly specified during system generation.

When nn is 32: For LIOCS:

1. An imperative macro (such as WRITE or PUT) was issued to a module that does not contain the requested function, or

2. A PUT was issued for an ISAM retrieve module without a preceding GET, or

3. An invalid ASA first character for the printer was used, or

4. A wrong length record indication occurred while processing 1287 documents when RECFORM=UNDEF, or

5. The 1287 program erroneously contained a CCW(s) with the SLI flag bit 'OFF', or

6. For COBOL, a wrong length record was detected in the object program.

When nn is any other value: The supervisor function requested by the operand of the SVC is not defined for the supervisor being used.

2.5.6.6 System Action Under Cancel. (Cont.)

- Cancel Code (Hexadecimal): 22

| Message Code: OS051 or OS061 |
|---|

Description, Action or Condition: This is caused by the issuing of a FETCH (SVC 1) or a LOAD (SVC 4) macro whose phase name cannot be found. This cancel code is also used when a logical transient is canceled.

- Cancel Code (Hexadecimal): 23

| Message Code: OS021 |
|---|

Description, Action or Condition: This is caused by a program, task or subtask issuing a CANCEL macro. If issued by a program or task, the program or partition is canceled. If issued by a subtask, the subtask only is canceled.

- Cancel Code (Hexadecimal): 24

| Message Code: OS1I |
|---|

Description, Action or Condition: This is a result of an operator entering CANCEL from the 1Ø52.

- Cancel Code (Hexadecimal): 25

| Message Code: OP71I |
|---|

Description, Action or Condition: This is a result of attempting to load a problem program phase at an address outside main storage or outside the requester's area (background or foreground). This condition also occurs:

If the program requires more main storage than is allocated to the partition where the program is to run, or if an improper address is detected on an SVC interrupt (i.e., CCW address in CCB is invalid).

2.5.6.6 System Action Under Cancel. (Cont.)

- Cancel Code (Hexadecimal): 26

| Message Code: OP71I |
|---|

Description, Action or Condition: This is a result of a program issuing an I/O request for a logical unit that is not assigned to a device. If a dump is taken, general register 1 contains the address of the CCB. If the CCB is unavailable, the logical unit message contains SYSxxx.

- Cancel Code (Hexadecimal): 27

| Message Code: OP70I |
|---|

Description, Action or Condition: This is a result of a program issuing an I/O request for a logical unit for which there is no logical unit block (LUB) entry (invalid LUB code in CCB). If a dump is taken, general register 1 contains the address of the CCB.

- Cancel Code (Hexadecimal): 28

| Message Code: None |
|---|

Description, Action or Condition: QTAM cancel in progress.

- Cancel Code (Hexadecimal): 3Ø

| Message Code: OP721 |
|---|

Description, Action or Condition: This is a result of a program ignoring the reading of the /& statement on SYSRDR or SYSIPT.

2.5.6.6 System Action Under Cancel. (Cont.)

- Cancel Code (Hexadecimal): 31

| Message Code: OP751 |
|---|

Description, Action or Condition: This is a result of the number of pending I/O errors exceeding supervisor capacity.

- Cancel Code (Hexadecimal): 32

| Message Code: OP76I |
|---|

Description, Action or Condition: This is caused by DASD file-protect limits being exceeded or by an incorrect record reference for system files on disk. It will also be posted for unrecoverable I/O errors on tape.

- Cancel Code (Hexadecimal): 33

| Message Code: OP79I |
|---|

Description, Action or Condition: This occurs when a DASD command chain in a file-protected environment does not start with a command code of X'Ø7'. This code indicates a long seek and must be the first command in the chain.

- Cancel Code (Hexadecimal): 34

| Message Code: OP841 |
|---|

Description, Action or Condition: This is caused by an unrecoverable I/O error during a FETCH of a non-$ phase, thus resulting in the job being canceled.

- Cancel Code (Hexadecimal): FF

2.5.6.6 System Action Under Cancel. (Cont.)

Message Code: OP78I

Description, Action or Condition: This occurred when an IBM-supplied component failed to post a valid cancel code.

All of these cancel codes cancel the program, task, or subtask when they occur. If multitasking is being used and a main task is canceled, all of the subtasks attached are detached and canceled as a result of the main task being canceled, with the exception of cancel code X'23'. If a dump option was specified at system generation time, the contents of the supervisor and the partition in which the cancel condition occurred is written on SYSLST.

The linkage editor map can be a great help in locating programs and subroutines that are included in the programs at object time. Common areas, load address, relation factors, low-core and high-core addresses are also shown. In addition, the PHASE card is displayed to show where the phase was loaded (i.e., directly following the supervisor or at some other location). This map is also helpful when working with multiphase programs.

The system dump of main storage used with these items allows the programer to relate all the information he has gathered to the contents of main storage at the time the error occurred. By using the dump and the listing, the programer can see how his program appeared in main storage at the time of the error. By using the values found in the Program Information Key (PIK) and Program Information Block (PIB) table in the dump, he can see partition save areas, registers, and instructions to determine what actually caused the error.

There are times when a system dump is not available to the programer, such as hard waits and unending loops. When one of these conditions occurs, the only way to get a dump of main storage is to use a stand-alone dump. Remember that the address of the communication region (COMRG) is lost when a stand-alone dump is taken. Therefore, bytes X'16' and X'17' should be displayed before taking a dump of main storage to ensure that the programer has the correct communication region address to use when he is analyzing the dump. If bytes '16' and '17' are not displayed, the communications region start address can still be found by scanning the dump for the date in the form MM/DD/YY or DD/MM/YY (this indicates the start of COMRG). Although the register values in a stand-alone dump (register print area of the dump) may not be valid, the partition save area values most likely will be valid.

LABEL STORAGE AREA (1 cylinder).

- The label storage area will start in track Ø of the first cylinder following the end of the SSL and will continue for one cylinder. Byte 13 of record 4 of the System Directory contains the address of the label storage cylinder. This address is also furnished for Job Control and LIOCS in the communication region.

2.5.6.6 System Action Under Cancel. (Cont.)

• The label storage cylinder will contain label blocks which define the labels on files used by processing programs. Label blocks are built at Job Control time for labeled tape files from // VOL and // TLBL cards and for disk files from // VOL, // DLBL and // EXTENT cards. These label blocks are retrieved from the label storage cylinder at execution time and read into main storage by the OPEN and CLOSE routines. File labels are also read by the OPEN and CLOSE routines into the transient area. A comparison is then made between the label block information and the file label to ensure that the proper file is being accessed.

• Core storage need not be reserved for sequential DASD files as the label blocks will be read into the transient area. Core storage must be reserved by the user for non-sequential DASD files and labeled tape files. However, core need be reserved for only the largest label block which will be retrieved. This core reservation is accomplished with the //LBLTYP statement at link edit time. This statement has two parameters: TAPE or NSD (nn).

• A // LBLTYP TAPE will reserve 80 bytes of user core and is used when ONLY labeled tape files are to be processed. This same 80 bytes is used by all tape files.

• A // LBLTYP NSD (nn) is used when non-sequential DASD files will be processed regardless of whether labeled tapes will also be processed. This statement reserves 84 bytes of core plus 20 bytes for each extent. A // LBLTYP NSD(nn) is submitted for the non-sequential DASD file with the largest number of extents with the number of extents being specified in the (nn) parameter. This same core storage will be used by label blocks for non-sequential DASD files with fewer extents and by labeled tape files.

• If neither non-sequential DASD label blocks nor labeled tape label blocks will be retrieved by the program, no core storage needs to be reserved (i.e., no // LBLTYP statement is submitted at link edit time).

• Only one // LBLTYP statement maximum is ever submitted at link edit time.

• LABELED TAPE LABEL BLOCKS.

  Always 80 bytes per file.

  Always read into lower portion of user core.

• LABEL BLOCK KEY FOR EVERY LABEL BLOCK. (Refer to FIGURE 2-41.)

| BYTE | FUNCTION | | CONTENT/FORMAT |
|---|---|---|---|
| 1-8 | File Name | CL8 | 'filename' |
| 9 | Reserved | X | '00' |
| 10 | Extent Sequence # | X | 'bb' |
| | | | A. Mag Tape 'bb'='00' |
| | | | B. DASD 'bb'=# of first extent |

FIGURE 2-41

2.5.6.6 System Action Under Cancel. (Cont.)

- LABEL BLOCKS SUMMARIZED. (Refer to FIGURE 2-42.)

| FILE TYPE | KEY | LABEL BLOCK |
|---|---|---|
| TAPE | 1Ø | 8Ø for each file |
| SEQ DASD | 1Ø | 1Ø4 for each extent |
| NON-SEQ DASD | 1Ø | 84 + 2Ø (number of extents) |

FIGURE 2-42

The label storage cylinder consists of two kinds of label blocks. Track Ø is used for permanent label blocks while tracks 1-9 are used for temporary label blocks. Permanent label blocks will remain in effect across job boundaries. Temporary label blocks are destroyed at job step completion time (end of // EXEC) by an EOF indication being placed in track 1 of the label storage cylinder.

Permanent label blocks for sequential disk files and labeled tape files can be created on track Ø by submitting a // OPTION STDLABEL statement prior to the // VOL, // DLBL and // EXTENT statements control cards. This procedure is recommended for sequential disk files used by the systems programs (SYSØØØ, SYSØØ1, SYSØØ2, SYSØØ3) so that // VOL, // DLBL and // EXTENT statements need not be submitted for every job. Permanent label blocks for all files must be rewritten every time a // OPTION STDLABEL control statement is used.

The format of the label blocks is dependent upon the kind of file being described as shown below.

- SEQUENTIAL DASD FILE LABEL BLOCKS. (Refer to FIGURE 2-43.)

1. Always at least 1Ø4 bytes per file.

2. Additional extents will require an additional 2Ø bytes similar to bytes 85-1Ø4.

3. Always read into lower portion of user core.

| BYTE | FUNCTION | | CONTENT/FORMAT |
|---|---|---|---|
| 1 | No. of Extents | X | 'bb'<br>A. SEQ DISK 'bb'='Ø1' for all except last extent.<br>B. SEQ DISK 'bb'=FF' for last extent.<br>C. Non-SEQ 'bb'=number of extents (maximum of 125) |

FIGURE 2-43

2.5.6.6 System Action Under Cancel. (Cont.)

| BYTE | FUNCTION | | CONTENT/FORMAT |
|---|---|---|---|
| 2-9 | File Name | CL8 | 'filename' (DTF name) |
| 10-53 | File ID | CL44 | 'Qualified name (Gen# Ver#)' |
| 54 | Format ID | CL1 | '1' |
| 55-60 | File Serial | CL6 | 'serial' |
| 61-62 | Volume Sequence | H | 'n' |
| 63 | Creation Date | X | 'yy' |
| 64-65 | " " | H | 'ddd' |
| 66 | Expiration Date | X | 'yy' |
| 67-68 | " " | H | 'ddd' |
| 69-71 | Reserved | 3X | '00' |
| 72-84 | System Code | CL13 | 'alphamerics' |
| 85-90 | Extent Serial | CL6 | 'serial' |
| 92 | Extent Seq. No. | X | 'bb'<br>A. ISFMS without a master index 'bb'='01'<br>B. All others 'bb'='00' |
| 93-96 | Lower Limit | 4X | 'C1', 'C2', 'H1', 'H2' |
| 97-100 | Upper Limit | 4X | 'C1', 'C2', 'H1', 'H2' |
| 101-102 | Symbolic unit from CCB | 2X | '0b', 'bb' |
| 103 | Old cell | X | 'B1' |
| 104 | New cell | X | 'B2' |

FIGURE 2-43 (Cont.)

2.5.6.7 Wait States.

- The system is said to be in a wait state when the "wait" light is continuously lit and the "system" light is off. Wait states are divided into hard waits and soft waits.

- If the system is in a hard wait, the wait bit in the current PSW (bit 14) is set to one and the system mask is set to zeros, thus disabling all interrupts. Because no interrupts are allowed, a PSW swap cannot occur and the system must be re-IPLed to continue processing.

- A soft wait occurs when the DOS supervisor finds no in-core programs ready to run and loads a PSW with the wait bit set to one and the system mask set to ones. The first interrupt returns control to the supervisor and processing continue.

- A wait can easily be determined as hard or soft by causing an interrupt. If the system responds with some action, the wait is soft; if not, the attention routine responds with the "READY FOR COMMUNICATIONS" message.

2.5.6.7 Wait States. (Cont.)

SOFT WAITS. If the system is in a continuous soft wait, it is waiting for an interrupt to signal the completion of an event. Although the expected interrupt may be from the timer or external interrupt key, a missing device-end caused by hardware is the most frequent cause. The operator can make each device not-ready, then ready, to generate a device-end interrupt from each address. The system light flashes briefly as the supervisor examines and discards interrupts for which it was not waiting. The interrupt from the device waited for causes normal processing to continue. (The occurrence should be brought to the attention of the customer engineer as a possible hardware failure.) If this technique does not end the wait, take a stand-alone dump to find what the system was waiting for.

HARD WAITS. The DOS supervisor loads a hard-wait PSW when a failure occurs that puts the integrity of the control program or system data in doubt. The supervisor attempts to place a message in low core bytes Ø-4.

• If a hard wait occurs, it is imperative that this message be retrieved and recorded. Effective diagnosis is extremely difficult if this step is neglected.

• If byte one of main storage contains an S (HEX 'E2'), the following information can be obtained easily:

• Check byte '73' for a 'OF'. This indicates either a channel control check or an interface control check. Bytes '3A'-'3B' contain the device address. If byte '73' does not contain a 'OF', a machine check must have occurred.

• Byte one may have a W. If a W (HEX 'E6') is found, a hard stop on SYSREC is indicated.

• If the CPU detects an error in its own circuitry, or (in the System/36Ø, model 5Ø or smaller) in the channel or interface control circuits, it forces a machine check interrupt. The system places an "S" in byte 1 and enters a hard wait. The "S" is a request to run the SEREP (System Environmental Recording, Editing, and Printing) dump to format and display the contents of the CPU's hardware registers and log-out area for use by the customer engineer. (A SEREP dump configured for the system should be available to the operator. A copy can be obtained from the customer engineer responsible for the CPU.)

2.5.6.8 Commonly Encountered User Errors.

• The following is a list of eight of the most commonly encountered type of user errors under the disk operating system (DOS). For each, the error condition is shown and possible reasons for the error given. Not all of the information provided will apply in every case, but as far as possible, common causes and sources of information are pointed out.

2.5.6.8 Commonly Encountered User Errors. (Cont.)

SPECIFICATION EXCEPTION.

- When data, an instruction or control-word address does not specify an integral boundary for the unit of information.

- When the multiplier or divisor in decimal arithmetic exceeds 15 digits and sign.

- When the first operand field is shorter than or equal to the second operand field in decimal multiplication or division.

OPERATOR INTERVENTION. The console operator will cancel a job when it is apparent the program is executing instructions in a manner that indicates a closed loop.

I/O EXCEPTION. This occurs when the program requests an illegal file handling operation. Reading, writing, etc.

DATA EXCEPTION. During arithmetic operations data is always in packed decimal form. The low order byte of each operand must contain a valid sign A-F.

OPERATION EXCEPTION. Occurs when a program is attempting to execute an illegal operation code. Can be caused by the forcing of control to outside the Procedure Division. Can be caused by a bad subscripting move overlaying instructions.

ADDRESSING EXCEPTION. Occurs when an instruction attempts to access an address outside of available storage. May be caused by:

- Program too large.
- Bad subscript.
- I/O malfunction.
- Bad register modification.

PROTECTION EXCEPTION. Occurs when an instruction attempts to access an address in some other partition. Can be caused by:

- Program too large.
- Bad subscript.

SUPERVISOR CALL ERRORS. Supervisor Call Errors, SVC's, are caused by a program requesting action from the supervisor with required prerequisite information either not available or improperly defined. This can be caused by

2.5.6.8 Commonly Encountered User Errors. (Cont.)

describing a file differently than the way it was created or giving other erroneous information concerning the file -- wrong record size, wrong blocking factor, wrong recording mode, wrong logical assignment. When the supervisor recognizes some deficiency, it will plug into the DTF for the file an '0A 32' to indicate the SVC 32 error.

2.5.6.9 DOS Core Dump Tracing.

- If a serious error occurs during execution of the problem program, the job is abnormally terminated; any remaining steps are bypassed; and a program phase dump is generated. The programer can use the dump for program checkout. (However, any pending transfers to an external device may not be completed. For example, if a READY TRACE statement is in effect when the job is abnormally terminated, the last card number may not appear on the external device.) In cases where a serious error occurs in other than the problem program (e.g., Supervisor), a dump is not produced. Note that program phase dumps can be suppressed if the NODUMP option of the OPTION control statement has been specified for the job, or if NODUMP was specified at system generation time and is not overridden by the DUMP option for the current job.

HOW TO USE A DUMP. When a job is abnormally terminated due to a serious error in the problem program, a message is written on SYSLST which indicates the:

- Type of interrupt (e.g., program check).
- Hexadecimal address of the instruction that caused the interrupt.
- Condition code.
- Reason for the interrupt (e.g., data exception).

The instruction address can be compared to the Procedure Division map.

The contents of LISTX provide a relative address for each statement. The load address of the module (which can be obtained from the map of main storage generated by the Linkage Editor) must be subtracted from the instruction address to obtain the relative instruction address as shown in the Procedure Division map. If the interrupt occurred within the COBOL program, the programer can use the error address and LISTX to locate the specific statement in the program which caused a dump to be taken. Examination of the statement and the fields associated with it may produce information as to the specific nature of the error.

In the following sample dump which was caused by a data exception, invalid data (i.e., data which did not correspond to its usage) was placed in the numeric field B as a result of redefinition. The following discussion illustrates the method of finding the specific statement in the program which caused the

2.5.6.9 DOS Core Dump Tracing. (Cont.)

dump. Letters identifying the text correspond to letters in the program listing. (See FIGURE 2-44.)

• The program interrupt occurred at HEX LOCATION 0039BC (Circle A). This is indicated in the SYSLST message printed just before the dump.

• The linkage editor map (Circle D) indicates that the program was loaded into address 0032A0. This is determined by examining the load point of the control section TESTRUN. TESTRUN is the name assigned to the program module by the source coding: PROGRAM-ID. TESTRUN.

• The specific instruction which caused the dump is located by subtracting the load address (Circle B) from the interrupt address (Circle A) (i.e., subtracting 32A0 from 39BC). The result, 71C, is the relative interrupt address and can be found in the object code listing (Circle C). In this case the instruction in question is an AP (add decimal).

• The left-hand column of the object code listing gives the compiler-generated card number associated with the instruction. It is card 69. As seen in the source listing, card 69 contains the COMPUTE statement.

LOCATING DATA.

• The location assigned to a given data-name may similarly be found by using the BL number and displacement given for that entry in the glossary, and then locating the appropriate one fullword BL cell in the TGT. The hexadecimal sum of the glossary displacement and the contents of the cell should give the relative address of the desired area. This can ther be converted to an absolute address as described above.

• Since the problem program interrupted because of a data exception, the programer should locate the contents of field B at the time of the interrupt. This can be done as follows:

• Locate data-name B in the glossary (Circle J). It appears under the column headed SOURCE-NAME. Source-name B has been assigned to base locator 3 (i.e., BL = 3) with a displacement of Ø5Ø. The sum of the value of base locator 3 and the displacement value 5Ø is the address of data-name B.

• The Register Assignment table (Circle K) lists the registers assigned to each base locator. Register 6 has been assigned to BL = 3.

• The contents of the sixteen general registers at the time of the interrupt are displayed at the beginning of the dump (Circle L). Register 6 contains the address 00003388.

2.5.6.9 DOS Core Dump Tracing. (Cont.)

- The location of data-name B can now be determined by adding the contents of register 6 and the displacement value 5Ø. The result 33D8, is the address of the leftmost byte of the 4-byte field B (Circle M).

NOTE: Field B contains F1F2F3C4. This is external decimal representation and does not correspond to the USAGE COMPUTATIONAL defined in the source listing.

2.5.6.9 DOS Core Dump Tracing. (Cont.)

```
                    ILLUSTRATION OF A PROGRAM PRODUCING
                    AN ABNORMAL TERMINATION WITH A DUMP

// JOB DTACHK                                                    05.00.19
// OPTION NODECK,LINK,LIST,LISTX,SYM,ERRS
// EXEC FCOBOL



 CBL QUOTE,SEQ
00001    000010 IDENTIFICATION DIVISION.
00002    000020 PROGRAM-ID. TESTRUN.
00003    000030     AUTHOR. PROGRAMMER NAME.
00004    000040     INSTALLATION. NEW YORK PROGRAMMING CENTER.
00005    000050     DATE-WRITTEN. FEBRUARY 4, 1971
00006    000060 DATE-COMPILED. 04/24/71
00007    000070    REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008    000080     COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
00009    000090     INPUT.
00010    000100
00011    000110 ENVIRONMENT DIVISION.
00012    000120 CONFIGURATION SECTION.
00013    000130 SOURCE-COMPUTER. IBM-360-H50.
00014    000140 OBJECT-COMPUTER. IBM-360-H50.
00015    000150 INPUT-OUTPUT SECTION.
00016    000160 FILE-CONTROL.
00017    000170     SELECT FILE-1 ASSIGN TO SYS008-UT-2400-S.
00018    000180     SELECT FILE-2 ASSIGN TO SYS008-UT-2400-S.
00019    000190
00020    000200 DATA DIVISION.
00021    000210 FILE SECTION.
00022    000220 FD  FILE-1
00023    000230     LABEL RECORDS ARE OMITTED
00024    000240     BLOCK CONTAINS 5 RECORDS
00025    000250     RECORDING MODE IS F
00026    000255     RECORD CONTAINS 20 CHARACTERS
00027    000260     DATA RECORD IS RECORD-1.
00028    000270 01  RECORD-1.
00029    000280     05 FIELD-A PIC X(20).
00030    000290 FD  FILE-2
00031    000300     LABEL RECORDS ARE OMITTED
00032    000310     BLOCK CONTAINS 5 RECORDS
00033    000320     RECORD CONTAINS 20 CHARACTERS
00034    000330     RECORDING MODE IS F
00035    000340     DATA RECORD IS RECORD-2.
00036    000350 01  RECORD-2.
00037    000360     05 FIELD-A PIC X(20).
```

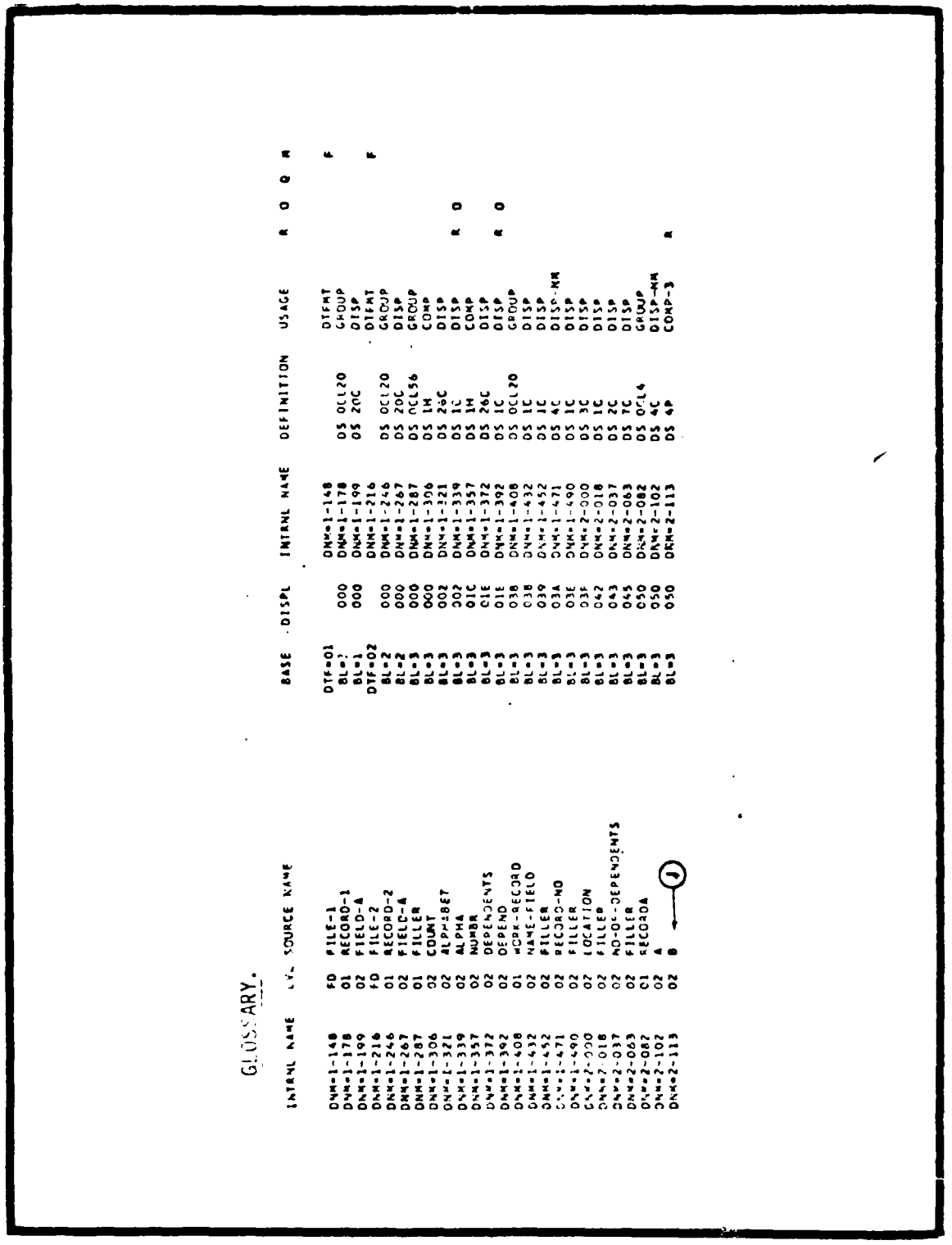
FIGURE 2-44

2.5.6.9 DOS Core Dump Tracing. (Cont.)

```
00038   000370 WORKING-STORAGE SECTION.
00039   000380 01  FILLER.
00040   000390     02 COUNT PIC S99 COMP SYNC.
00041   000400     02 ALPHABET PIC X(26) VALUE IS "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
00042   000410     02 ALPHA REDEFINES ALPHABET PIC X OCCURS 26 TIMES.
00043   000420     02 NUMBR PIC S99 COMP SYNC.
00044   000430     02 DEPENDENTS PIC X(26) VALUE "01234012340123401234012340".
00045   000440     02 DEPEND REDEFINES DEPENDENTS PIC X OCCURS 26 TIMES.
00046   000450 01  WORK-RECORD.
00047   000460     05 NAME-FIELD PIC X.
00048   000470     05 FILLER PIC X.
00049   000480     05 RECORD-NO PIC 9999.
00050   000490     05 FILLER PIC X VALUE IS SPACE.
00051   000500     05 LOCATION PIC AAA VALUE  S "NYC".
00052   000510     05 FILLER PIC X VALUE IS SPACE.
00053   000520     05 NO-OF-DEPENDENTS PIC XX.
00054   000530     05 FILLER PIC X(7) VALUE IS SPACES.
00055   000534 01  RECORDA.
00056   000535     02 A PICTURE S9(4) VALUE 1234.
00057   000536     02 B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
00058   000540
00059   000550 PROCEDURE DIVISION.
00060   000560 BEGIN. READY TRACE.
00061   000570     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00062   000580     AND INITIALIZES COUNTERS.
00063   000590 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO COUNT, NUMBR.
00064   000600     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
00065   000610     CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
00066   000620     THEM ON THE CONSOLE.
00067   000630 STEP-2. ADD 1 TO COUNT, NUMBR. MOVE ALPHA (COUNT) TO
00068   000640     NAME-FIELD.
00069   000645         COMPUTE B = B + 1.                    (D)
00070   000650     MOVE DEPEND (COUNT) TO NO-OF-DEPENDENTS.
00071   000660     MOVE NUMBR TO RECORD-NO.
00072   000670 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
00073   000680     WORK-RECORD.
00074   000690 STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL COUNT IS EQUAL TO 26.
00075   000700     NOTE THAT THE FOLLOWING CLOSES THE OUTPUT FILE AND REOPENS
00076   000710     IT AS INPUT.
00077   000720 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00078   000730     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES
00079   000740     OUT EMPLOYEES WITH NO DEPENDENTS.
00080   000750 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00081   000760 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00082   000770     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO STEP-6.
00083   000780 STEP-8. CLOSE FILE-2.
00084   000790     STOP RUN.
```

FIGURE 2-44 (Cont.)

2.5.6.9 DOS Core Dump Tracing. (Cont.)



GLOSSARY.

| INTRNL NAME | LVL | SOURCE NAME | BASE | DISPL | INTRNL NAME | DEFINITION | USAGE | R | O | Q | M |
|---|---|---|---|---|---|---|---|---|---|---|---|
| DNM=1-148 | FD | FILE-1 | DTF=01 | | DNM=1-148 | | DTFMT | | | | F |
| DNM=1-178 | 01 | RECORD-1 | BL=? | 000 | DNM=1-178 | DS 0CL20 | GROUP | | | | |
| DNM=1-199 | 02 | FIELD-A | BL=1 | 000 | DNM=1-199 | DS 20C | DISP | | | | |
| DNM=1-216 | FD | FILE-2 | DTF=02 | | DNM=1-216 | | DTFMT | | | | F |
| DNM=1-246 | 01 | RECORD-2 | BL=2 | 000 | DNM=1-246 | DS 0CL20 | GROUP | | | | |
| DNM=1-267 | 02 | FIELD-A | BL=2 | 000 | DNM=1-267 | DS 20C | DISP | | | | |
| DNM=1-287 | 01 | FILLER | BL=3 | 000 | DNM=1-287 | DS 0CL56 | GROUP | | | | |
| DNM=1-306 | 02 | COUNT | BL=3 | 000 | DNM=1-306 | DS 1H | COMP | | | | |
| DNM=1-321 | 02 | ALPHABET | BL=3 | 002 | DNM=1-321 | DS 26C | DISP | | | | |
| DNM=1-339 | 02 | ALPHA | BL=3 | 002 | DNM=1-339 | DS 1C | DISP | R | O | | |
| DNM=1-357 | 02 | NUMBR | BL=3 | 01C | DNM=1-357 | DS 1H | COMP | | | | |
| DNM=1-372 | 02 | DEPENDENTS | BL=3 | 01E | DNM=1-372 | DS 26C | DISP | | | | |
| DNM=1-392 | 02 | DEPEND | BL=3 | 01E | DNM=1-392 | DS 1C | DISP | R | O | | |
| DNM=1-408 | 01 | WORK-RECORD | BL=3 | 038 | DNM=1-408 | DS 0CL20 | GROUP | | | | |
| DNM=1-432 | 02 | NAME-FIELD | BL=3 | 038 | DNM=1-432 | DS 1C | DISP | | | | |
| DNM=1-452 | 02 | FILLER | BL=3 | 039 | DNM=1-452 | DS 1C | DISP | | | | |
| DNM=1-471 | 02 | RECORD-NO | BL=3 | 03A | DNM=1-471 | DS 4C | DISP-NM | | | | |
| DNM=1-490 | 02 | FILLER | BL=3 | 03E | DNM=1-490 | DS 1C | DISP | | | | |
| DNM=2-000 | 02 | LOCATION | BL=3 | 03F | DNM=2-000 | DS 3C | DISP | | | | |
| DNM=2-018 | 02 | FILLER | BL=3 | 042 | DNM=2-018 | DS 1C | DISP | | | | |
| DNM=2-037 | 02 | NO-OF-DEPENDENTS | BL=3 | 043 | DNM=2-037 | DS 2C | DISP | | | | |
| DNM=2-063 | 02 | FILLER | BL=3 | 045 | DNM=2-063 | DS 7C | DISP | | | | |
| DNM=2-082 | 01 | RECORD4 | BL=3 | 050 | DNM=2-082 | DS 0CL4 | GROUP | | | | |
| DNM=2-102 | 02 | A | BL=3 | 050 | DNM=2-102 | DS 4C | DISP-NM | | | | |
| DNM=2-113 | 02 | B | BL=3 | 050 | DNM=2-113 | DS 4P | COMP-3 | | | | A |

FIGURE 2-44 (Cont.)

2.5.6.9 DOS Core Dump Tracing. (Cont.)

MEMORY MAP

| | |
|---|---|
| TGT | 00358 |
| SAVE AREA | 003E8 |
| SWITCH | 00430 |
| TALLY | 00434 |
| SORT SAVE | 00438 |
| ENTRY-SAVE | 0043C |
| SORT CORE SIZE | 00440 |
| NSTD-REELS | 00444 |
| SORT RET | 00446 |
| WORKING CELLS | 00448 |
| SORT FILE SIZE | 00578 |
| SORT NODE SIZE | 0057C |
| PGT-VN TBL | 00580 |
| TGT-VN TBL | 00584 |
| SORTAB ADDRESS | 00588 |
| LENGTH OF VN TBL | 0058C |
| LNGTH OF SORTAB | 0058E |
| PGM ID | 00590 |
| A(INITI) | 00598 |
| UPSI SWITCHES | 0059C |
| OVERFLOW CELLS | 005A4 |
| BL CELLS | 005A4 |
| DTFADR CELLS | 00580 |
| TEMP STORAGE | 005B8 |
| TEMP STORAGE-2 | 005C0 |
| TEMP STORAGE-3 | 005C0 |
| TEMP STORAGE-4 | 005C0 |
| BLL CELLS | 005C0 |
| VLC CELLS | 005C4 |
| SBL CELLS | 005C4 |
| INDEX CELLS | 005C4 |
| SUBADR CELLS | 005C4 |
| ONCTL CELLS | 005CC |
| PFMCTL CELLS | 005CC |
| PRNSAV CELLS | 005CC |
| VN CELLS | 005D0 |
| SAVE AREA #2 | 005D4 |
| ASASM CELLS | 005D4 |
| XSA CELLS | 005D4 |
| PARAM CELLS | 005D4 |
| RPTSAV AREA | 005D8 |
| CHECKPT CTR | 005D8 |
| TOPTR CELLS | 005D8 |

FIGURE 2-44 (Cont.)

2.5.6.9 DOS Core Dump Tracing. (Cont.)

```
REGISTER ASSIGNMENT

 REG 6   BL =3   <-(A)
 REG 7   BL =1
 REG 8   BL =2

OBJECT CODE LISTING.

67                000006FC  41 40 6 002              LA    4,002(0,6)          DNR=1-339
                  00070O    48 20 6 000              LH    2,000(0,6)          DNR=1-306
                  000704    4C 20 C 03A              MH    2,03A(0,12)         LIT+2
                  000708    1A 42                    AR    4,2
                  00070A    58 40 C 038              S     4,038(0,12)         LIT+0
                  00070E    50 4C D 1DC              ST    4,1DC(0,13)         SBS+1
                  000712    58 E0 D 1DC              L     14,1DC(0,13)        SBS+1
                  000716    D2 00 6 038 E 000        MVC   038(1,6),000(14)    DNR=1-432      DNR=1-339
69                00071C    FA 30 6 050 C 03C  (C)-> AP    050(4,6),03C(1,12)  DNR=2-113      LIT+4
70                000722    41 40 6 01E              LA    4,01E(0,6)          DNR=1-392
                  000726    48 20 6 000              LH    2,000(0,6)          DNR=1-306
                  00072A    4C 20 C 03A              MH    2,03A(0,12)         LIT+2
                  00072E    1A 42                    AR    4,2
                  000730    58 40 C 038              S     4,038(0,12)         LIT+0
                  000734    50 40 D 1E0              ST    4,1E0(0,13)         SBS+2
                  000738    58 E0 D 1E0              L     14,1E0(0,13)        SBS+2
                  00073C    D2 00 6 043 E 000        MVC   043(1,6),000(14)    DNR=2-37       DNR=1-392
                  000742    92 40 6 044              MVI   044(6),X'40'        DNR=2-37+1

// EXEC LNKEDT

LINKAGE EDITOR MAP.

        PHASE   XFR-AD  LOCORE  HICORE  DSK-AD   ESD TYPE  LABEL     LOADED  REL-FR
                                                                                  <-(B)
      PHASE***  0032A0  003240  004ADB  53 D1 2  CSECT     TESTRUN   0032A0  0032A0

                                                 CSECT     IJFFBZZN  003C50  003C50
                                                 * ENTRY   IJFFZZZN  003C50
                                                 * ENTRY   IJFFBZZZ  003C50
                                                 * ENTRY   IJFFZZZZ  003C50

                                                 CSECT     ILBDSAE0  0049F0  0049F0
                                                    ENTRY  ILBDSAE1  004A06

                                                 CSECT     ILBDMNS0  0049E8  0049E8

                                                 CSECT     ILBDDSP0  004188  004188
                                                 * ENTRY   ILBDDSP1  004708
                                                 * ENTRY   ILBDDSP2  0047A0
                                                 * ENTRY   ILBDDSP3  004958

                                                 CSECT     ILBDIML0  004990  004990

                                                 CSECT     IJJCPD1   003FC0  003FC0
                                                    ENTRY  IJJCPDIN  003FC0
                                                 * ENTRY   IJJCPD3   003FC0

// ASSGN SYS006,X'182'
// EXEC
```

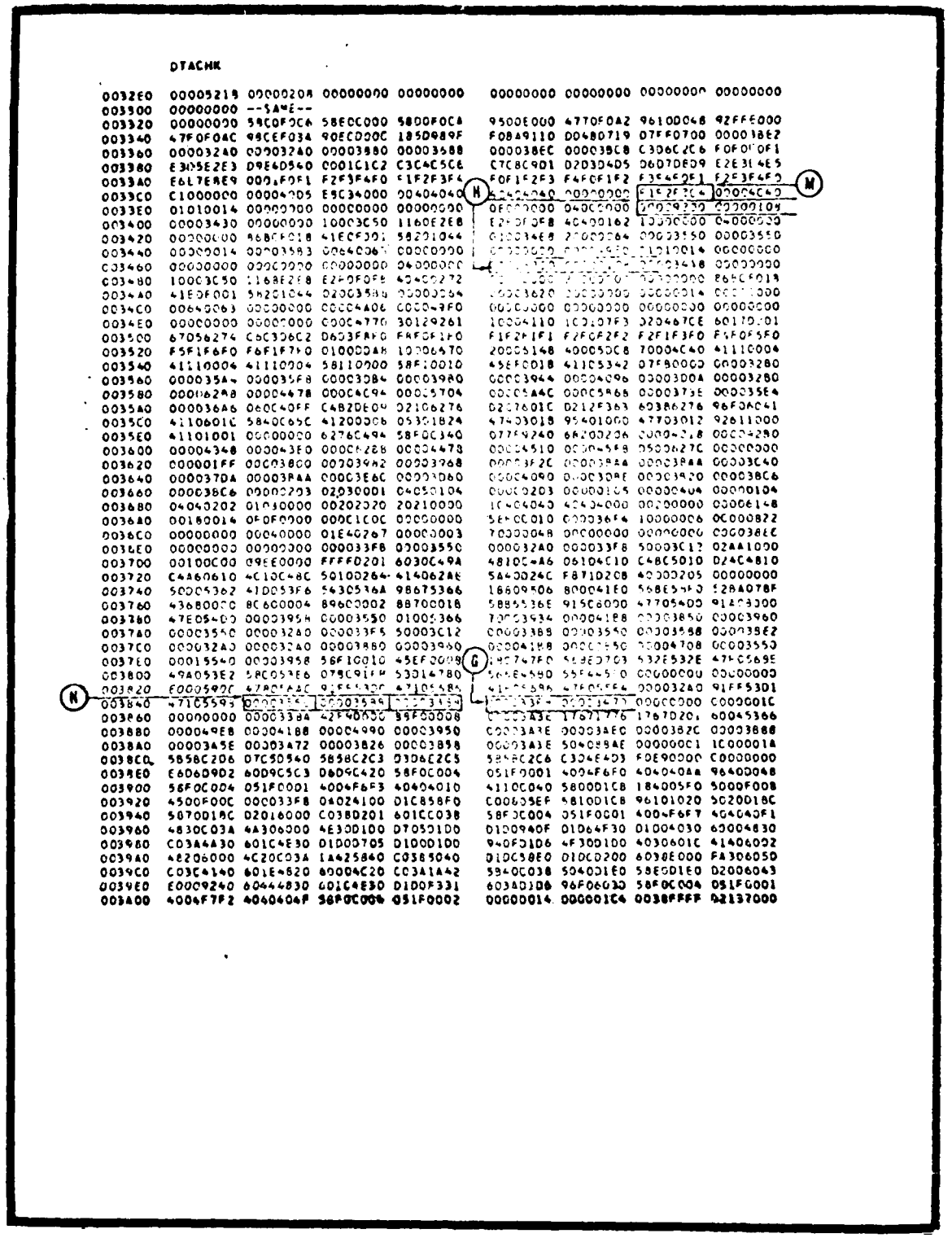
FIGURE 2-44 (Cont.)

## 2.5.6.9 DOS Core Dump Tracing. (Cont.)

```
05031 PROGRAM CHECK INTERRUPTION - HEX LOCATION 003B8C - CONDITION CODE 0 - DATA EXCEPTION   (A)
05001 JOB DTACHR   CANCELED


        DTACHR                                                                    (1)

GR 0-7  00003850 00003960 00000001 00000001   0000338A 50003C12 00003388 00003550
GR 8-F  00003588 000038E2 000032A0 000032A0   00003880 00003688 0000338A 00004188
FP REG  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000
COMREG  BG ADDR IS 000208


000000  00000000 00000000 00000000 00000000   00000000 00000208 FFFC0000 00000000
000020  FF050007 40002E06 FF150007 C0003AC2   56540205 0401F440 FF050006 40002E00
000040  00002F28 08000000 00002F16 00000003   FCB81CB3 01503588 00040006 0F001484
000060  00040000 00000338 00040000 0001047A   00000000 00003B8C 00040000 00000204
000080  00000003 00000000 00000003 00003003   00050003 06B006B0 06B04168 00734570
0000A0  0146940F B4784140 C0544570 08841AB   41900156 41A2B2CE 47F0000A 06B00660
0000C0  06B00680 06B00680 06B04188 001741B8   00504570 01464180 01569640 A0019120
0000E0  A00C4710 00EA9260 40019562 A0024780   00C695C1 A0024780 00CA9561 A0024780
000100  010E9104 A0004780 010E9203 005F9281   A0004BA0 026248FA C000494A 027641AA
000120  C0440770 94F97038 07017058 70589283   4C005680 A0014400 04080788 947FA001
000140  45708218 07FB42B0 00E74980 02CB47F0   BC7C4570 BC7DD205 BEEEBEF5 0C058EEE
000160  C04418AA D006BEFE 000C4341 000742A0   023741AA C0444400 A0045890 A0044220
000180  A0009140 A0014710 B364D207 01F09008   68009058 6B209060 68409068 68609070
0001A0  4BA00262 41AAC000 02010016 A0009898   90108200 01FC4400 A0045890 A0049818
0001C0  90309890 01F08200 00389284 C0A4D207   01F0BF50 9590BF58 A2001FF0 968C4000
0001E0  41100030 47F08166 96030039 82000038   FF050007 40002E06 00001000 00002000
000200  000C3022 80001048 FCF461F2 F461F7F1   32403000 00000000 02000000 00000000
000220  C4F3C1C3 C8024040 0007AFFF 00004A08   00004A08 00000016 0007AFFF F875ECD1
000240  A8407C00 00C62171 21782269 226A0000   25102514 2518BCF0 1AF2F4F7 F1F1F1F4
000260  00002046 00C00000 22E21E4E 15F41F04   1F140020 214C0010 5858C206 0C130001
000280  01001F98 20001C00 00000000 02080000   00000294 00000000 00026540 00C00044
0002A0  00001F2C 00000000 00000000 00000000   00000000 00000000 00000000 00000000
0002C0  0006289C 00003228 1000200E 000010C8   0000249C 92380104 40900160 41900696
0002E0  48A00236 48A00262 9180A000 47100306   5880A004 90188030 48B002CB 41CB8000
000300  41DCB000 07F99601 A0004480 02CB41CB   B0004104 B0009558 A00F0789 90E0BF6C
000320  48E001C8 0207BF50 E0039450 BF510213   BF5801F0 07F99090 01F09220 01C94590
000340  02E04190 01669500 00234780 03F49526   00234780 00BE4860 00221A6A 487002C8
000360  48667000 07F61818 18664121 D03F4570   BCAC4860 BE5C1822 43201007 41300018
000380  18234740 03904130 00151823 47800392   1A234220 04094320 100747F0 04584720
0003A0  00C44230 04094820 02364322 C003A423   95081007 47F00454 47200008 96801302
0003C0  960C1004 07F91858 41403002 43540000   41455000 1A444440 BE5495FF 43004770
0003E0  03CC4284 00300759 95FFC481 07891800   50008A74 95FF0481 47809238 48600236
000400  95600237 47800366 05024005 02814770   04204111 00304910 BE3447F0 04201866
000420  41210008 457CBCAC D5021009 02394780   00C61833 43301007 95011006 47700398
000440  92FF0409 02000440 A00D4123 000BD530   1007A00E 47800CCA D6011002 C0954182
000460  20004870 02544138 70004930 BE384780   03868930 00034430 02439160 30944780
000480  860ED501 02229B2C 47808BE4 D5018E4C   BE544720 04AE4930 BE744770 04AE950F
0004A0  00234750 04AE9110 30064780 B2384180   00014148 800C1844 44408E54 18584850
0004C0  02740200 04914000 50104000 92FF4000   42205000 42605000 92035018 91F03304
0004E0  47805628 47F004EC 58808F8C 44000C88   9560C09C 47700566 02020200 10095860
000500  02CC9507 60004770 0566D202 02CD6001   587002CC 1B444340 3005C40 BE485440
000520  02004144 00500503 70014000 47800566   91201000 47100560 91051002 47700560
000540  91406004 47800560 94BF6004 91101002   47800558 96601002 96161002 9601130C
000560  02034000 70019588 30024770 03C64280   30029198 30060779 43203000 4322C09D
000580  48603000 95003030 47800592 9F006000   07694060 05E29550 10044780 080C9504
0005A0  50184780 08409101 100C4710 05D0940F   06FF91F0 500C4780 05500300 C09C3004
0005C0  95035018 47000634 9560C09C 4780067C   02020049 10099408 07030300 00485000
0005E0  9C000008 47700572 40320084 96803006   07E94730 08C49106 00454770 0E9C913F
000600  00454770 060C91AE 00440789 D2010038   05E29550 10044770 06200202 00491009
000620  58600042 4A608D0C 50600040 40320084   47F0059A 95015018 47200656 9560C09C
000640  47700654 45700894 91208008 47100096   47F0065C 91203006 47100094 9560C09C
000660  47700500 45700884 48600262 4878C000   91407038 47100500 96F006FF 95003003
```

FIGURE 2-44 (Cont.)

## 2.5.6.9 DOS Core Dump Tracing. (Cont.)



```
          DTACHK
003200  00005219 00000208 00000000 00000000   00000000 00000000 00000000 00000000
003300  00000000 --SAVE--
003320  00000000 59C0F0C4 58E0C000 5800F0C8   9500E000 4770F0A2 96100048 92FFE000
003340  47F0F04C 98CEF034 90ECD00C 185D989F   F08A9110 D0480719 07FE0700 00003BE2
003360  00003240 00003240 00003980 00003688   00003BEC 00003BC8 C3D6C2C6 F0F0F0F1
003380  E3D5E2E3 D9E4D540 C001C1C2 C3C4C5C6   C7C8C9D1 D2D3D4D5 D6D7D8D9 E2E3E4E5
0033A0  E6E7E8E9 00014F0F1 F2F3F4F0 F1F2F3F4   F0F1F2F3 F4F0F1F2 F3F4F0F1 F2F3F4F5
0033C0  C1000000 00004005 E9C34000 00404040   40404040 00000000 F1F2F3C4 00004C40
0033E0  01010014 00000700 000C0000 00000000   0F000000 040C0000 00000230 00000104
003400  00003430 00000000 100C3C50 1160E2E8   E2F0F0F8 40400162 10000000 04000000
003420  00000000 96BCF018 41E0F001 58201044   01003AE8 20000264 00003550 00003550
003440  00000014 00003583 0064C064 00000000   00000020 00014550 01210014 00000000
003460  00000000 00000000 00000000 04000000   41000000 00000000 00003418 00000000
003480  1000C3C50 1168E2E8 E2F0F0F8 40400272   00000000 04000000 00000000 B6BCF018
0034A0  41E0F001 58201044 0200C588 00000264   00003620 00000000 00000014 00000000
0034C0  00640063 00000000 C000C4A06 C000C3F0   00000000 00000000 00000000 00000000
0034E0  00000000 00007000 C000C4770 30129261   10004110 1001D7F3 32D467CE 6017D201
003500  67056274 C6C3D6C2 D6D3F4F0 F4F0F1F0   F1F2F1F1 F2F0F2F2 F2F1F3F0 F4F0F5F0
003520  F5F1F6F0 F6F1F7F0 01000DA8 10006470   20006148 40005DC8 70004C40 41110004
003540  41110004 41110004 58110000 58F10010   45EF0018 411C5342 07FB0000 00003280
003560  000035A4 000035F8 00003B84 00003980   00003944 00004096 00003D0A 00003280
003580  00003628 00004478 000C4C94 00005704   00005A4C 000C5968 00003736 000035E4
0035A0  000036A6 060C40FF C4B2DE04 02106276   D2176010 D212F363 60386276 96F0AC41
0035C0  41106010 5840C65C 41200D08 05301824   47403018 95401000 47703012 92611000
0035E0  41101001 00000000 6276C494 58F0C340   07FE9240 66200226 00004218 00004280
003600  00004348 00004EF0 0000C228 00004478   00004510 00004588 05000627 00000000
003620  000001FF 00003800 0070394A 00003768   00003F2C 00003F4A 00003F4A 00003C40
003640  00003704 00003FAA 000C3E6C 00003D60   00004090 00003DFE 00003920 00003BC6
003660  00003BC6 00002203 02030001 04050104   00002203 00000105 00000404 00000104
003680  04040202 01030000 00202320 20210000   10404043 40434000 00200000 00006148
0036A0  00180014 0F0F0000 000C1C0C 00000000   5E4C0010 00003614 10000006 0C000822
0036C0  00000000 00040000 01E4G2G7 00003003   70000048 00C00000 00000006 C0G038EC
0036E0  00000000 00000300 000033F8 03003550   00003240 000033F8 50003C12 02AA1000
003700  00100C00 09EE0000 FFFFD201 6030C49A   4810C4A6 0610C410 C48C5010 D24C4810
003720  C4A60610 4C10C48C 50100264 414062AE   5A40024C F8710208 40300205 00000000
003740  50005362 41D053F6 54305364 98675366   16609506 800041E0 568E5450 52BA078F
003760  43680000 8C600004 896C0002 88700018   58865366 915C8000 47705400 91A03000
003780  47E05400 00003954 00003550 01005366   70003534 00004188 00003850 00003960
0037A0  00003550 00003240 000033F8 50003C12   00003388 00003550 00003588 00003BE2
0037C0  00003240 00003240 00003880 00003960   00004188 00003E50 00004708 0C003550
0037E0  00015540 00003958 56F10010 45EF2018   [illegible] 5A3ED703 532E532E 47F05695
0037F0  49A053E2 58C053E6 07FC91FF 53314780   5A4E4580 55E44510 00000000 00000003
003820  F0005920 47F0F4AC 91FF5320 47105586   41005896 47F05FE4 00003240 91FF53D1
003840  47105594 00003540 00003594 00003A40   [illegible] 00003477 000C0000 C000001C
003860  00000000 00003384 42F0D03C 38F00008   00003A3E 17671776 1767D20, 60045366
003880  00004EB8 00004188 00004990 00003950   00003A3E 00003AEC 00003B2C 02003888
0038A0  00003A5E 00003A72 00003826 00003858   00003A3E 5040E9AE 00000001 1C000018
0038C0  5858C2D6 D7C5D540 5858C2C3 D3D6E2C5   5858C2C6 C3D4E4D3 F0E90000 C0000000
0038E0  E6D6D9D2 60D9C5C3 D6D9C420 58F0C004   051F0001 4004F6F0 4040404A 96400048
003900  58F0C004 051F0001 4004F6F3 40404010   4110C040 580001C8 184005F0 5000F008
003920  4500F00C 000033F8 04024100 D1C858F0   C0005EF 58100108 96101020 5020D18C
003940  5070D18C D2016000 C038D201 601CC038   58F3C004 051F0001 4004F6F7 4040404F1
003960  4830C03A 4A306000 4E30D100 D7051D00   D1D0940F D1D64F30 D1D04030 60004830
003980  C03A4A30 601C4E30 D1D00705 D1D01D00   940FD1D6 4F30D1D0 40306010 41406002
0039A0  48206000 4C20C03A 1A425840 C0385040   D1DC58E0 D1DCD200 6038E000 FA306050
0039C0  C03C4140 601E4820 60004C20 C03A1A42   5840C038 5040D1E0 58E0D1E0 D2006043
0039E0  E0009240 60444830 601C4E30 D1D0F331   603AD1D6 96F06030 58F0C004 051F0001
003A00  4004F7F2 40404040 58F0C004 051F0002   00000014 00000104 0038FFFF 02137000
```

FIGURE 2-44 (Cont.)

2.5.6.10 Interpreting Output.

• The Full American National Standard COBOL compiler, COBOL object module, Linkage Editor, and other system components can produce output in the form of printed listings, punched card decks, diagnostic or informative messages, and data files directed to tape or to mass storage devices. This chapter gives the format of and describes this output. The same COBOL program is used for each example. "Sample Program Output" shows the output formats in the context of a complete listing generated by the sample program.

COMPILER OUTPUT. The output of the compilation job step may include:

- A printed listing of the job control statements.
- A printed listing of the statements contained in the source program.
- A glossary of compiler-generated information about data.
- Global tables, register assignments, and literal pools.
- A printed listing of the object code.
- A condensed listing containing only the relative address of the first generated instruction for each verb.
- Compiler diagnostic messages.
- A cross-reference listing.
- System messages.
- An object module.

The presence or absence of the above-mentioned types of compiler output is determined by options specified at system generation time. These options can be overridden or additional options specified at compilation time by using the OPTION control statement and the CBL card.

The level of diagnostic message printed depends upon the FLAGW or FLAGE option of the CBL card.

All output to be listed is written on the device assigned to SYSLST. If SYSLST is assigned to a magnetic tape, COBOL will treat the file as an unlabeled tape. Line spacing of the source listing is controlled by the SPACEn option of the CBL card and by SKIP 1/2/3 and EJECT in the COBOL source program. The number of lines per page can be specified in the SET command. In addition, a listing of input/output assignments can be printed on SYSLST by using the LISTIO control statement.

2.5.6.10 Interpreting Output. (Cont.)

The illustration of compiler output (FIGURE 2-49) contains the compiler output listing showing Sample Program Output. Each type of output is numbered, and each format within each type is lettered. The text below and that following the listing is an explanation of the ENTRY.

- (Circle 1) The listing of the job control statements associated with this job step. These statements are listed because the LOG option was specified at system generation time.

- (Circle 2) Compiler options. The CBL card, if specified, is printed on SYSLST unless the LIST option is suppressed.

- (Circle 3) The source module listing. The statements in the source program are listed exactly as submitted except that a compiler-generated card number is listed to the left of each line. This is the number referenced in diagnostic messages and in the object code listing. It is also the number printed on SYSLST as a result of the source language TRACE statement. The source module is not listed when the NOLIST option is specified.

- The following notations may appear on the listing:

  "C" Denotes that the statement was inserted with a COPY statement.

  ** Denotes that the card is out of sequence. NOSEQ should be specified on the CBL card if the sequence check is to be suppressed.

  "I" Denotes that the card was inserted with an INSERT or BASIS card.

- If DATE-COMPILED is specified in the Identification Division, any sentences in that paragraph are replaced in the listing by the date of compilation. It is printed in one of the following formats depending upon the format chosen at system generation time.

```
DATE-COMPILED.  month/day/year  or
DATE-COMPILED.  day/month/year
```

(Circle 4) Glossary. The glossary is listed when the SYM option is specified. The glossary contains information about names in the COBOL source program.

2.5.6.10 Interpreting Output. (Cont.)

Columns 'A' and 'F', the internal-name generated by the compiler. This name is used in the compiler object code listing to represent the name used in the source program. It is repeated in column 'F' for readability.

Column 'B', a normalized level number. This level number is determined by the compiler as follows: the first level number of any hierarchy is always Ø1, and increments for other levels are always by one. Only level numbers Ø3 through 49 are affected; level numbers 66 (not USACSC COBOL), 77, and 88, and FD, SD, RD indicators are not changed.

Column 'C', the data-name that is used in the source module.

Columns 'D' and 'E', for data-names, contain information about the address in the form of a base and displacement. For file-names, the column contains information about the associated DTF, if any.

Column 'G', defines storage for each data item. It is represented in assembler-like terminology. The Glossary definition and usage table refers to information in this column.

Column 'H', usage of the data-name. For FD entries, the DTF type is identified (e.g., DTFDA). For group items containing a USAGE clause, the usage type is printed. For group items that do not contain a USAGE clause, GROUP is printed. For elementary items, the information in the USAGE clause is printed.

GLOSSARY DEFINITION AND USAGE. (Refer to FIGURE 2-45.)

2.5.6.10 Interpreting Output. (Cont.)

| TYPE | DEFINITION | USAGE |
|---|---|---|
| Group-Fixed-Length | DS OCLN | GROUP |
| Alphabetic | DS NC | DISP |
| Alphanumeric | DS NC | DISP |
| Alphanumeric Edited | DS NC | AN-EDIT |
| Numeric Edited | DS NC | NM-EDIT |
| Index-Name | DS 1H | INDEX-NM |
| Group Variable-Length | DS-VLI-N | GROUP |
| Sterling Report | DS NC | RPT-ST |
| External Decimal | DS-NC | DISP-NM |
| External Floating-Point | DS-NC | DISP-FP |
| Internal Floating-Point | DS 1F | COMP-1 |
| | DS 1D | COMP-2 |
| Binary | DS 1H, 1F, or 2F | COMP |
| Internal Decimal | DS NP | COMP-3 |
| Sterling Non-Report | DS NC | DISP-ST |
| Index-Name | BLANK | INDEX-NAME |
| File (FD) | BLANK | DTF TYPE |
| Condition (88) | BLANK | BLANK |
| Report Definition (RD) | BLANK | BLANK |
| Sort Definition (SD) | BLANK | BLANK |

Note: Under the definition column, N=size in bytes, except in group variable-length where it is a variable cell number.

FIGURE 2-45

2.5.6.10 Interpreting Output. (Cont.)

Column 'J', the letter under column:

R - Indicates that the data-name redefines another data-name.

O - Indicates that an OCCURS clause has been specified for that data-name.

Q - Indicates that the data-name is or contains the DEPENDING ON object of the OCCURS clause.

M - Indicates the record format. The letters which may appear under column M are:

F - fixed-length records.

U - undefined records.

V - variable-length records.

S - spanned records.

(Circle 5) Global tables and literal pool are listed when the LISTX option is specified, unless SUPMAP is also specified and an E-level error is encountered. A global table contains easily addressable information needed by the object program for execution. For example, in the Procedure Division output coding (3), the address of the first instruction under STEP-1 (OPEN OUTPUT FILE-1) is found in the PROCEDURE NAME CELLS portion of the Program Global Table ($PG^T$).

(Circle (5,A)) The Task Global Table (TGT) is used to record and save information needed during the execution of the object program. This information includes switches, addresses, and work areas.

(Circle (5,B)) The Literal Pool lists all literals used in the program, with duplications removed. These literals include those specified by the compiler (e.g., to align decimal points in arithmetic computations). The literals

2.5.6.10 Interpreting Output. (Cont.)

are divided into two groups: those that are referenced by instructions (marked "LITERAL POOL") and those that are parameters to the display object time subroutine (marked "DISPLAY LITERALS").

(Circle (5,C)) The Program Global Table (PGT), contains literals and the addresses of procedure-names and generated procedure-names referenced by Procedure Division irstructions.

(Circle (5,6)) Register assignment lists the permanent register assigned to each base locator in the object program. The remaining base locators are given temporary register assignments but are not listed. Register assignments are listed when LISTX is specified.

(Circle 7) The object code listing is produced when the LISTX option is specified, unless SUPMAP is also specified and an E-level error is encountered. The actual object code listing contains:

(Column A) The compiler-generated card number identifies the COBOL statement in the source deck which contains the verb that generates the object code found in column C.

(Column B) The relative location, in hexadecimal notation, of the object code instruction in the module.

(Column C) The actual object code instruction in hexadecimal notation.

(Column D) The procedure-name number. A number is assigned only to procedure-names referred to in other Procedure Division statements.

(Column E) The object code instruction in the form that closely resembles assembler language (Displacements are in hexadecimal notation).

(Column F) Compiler-generated information about the operands of the generated instruction. This includes names and relative locations of literals.

(Circle 8) The cross reference dictionary is produced when the XREF option is specified. It consists of two parts:

(Column A) The XREF dictionary for data-names consists of data-names followed by the generated card number of the statement which defines each data-name, and the generated card number of statements where each data-name is referenced.

(Column B) The XREF dictionary for procedure-names consists of the procedure-names followed by the generated card number of the statement where each procedure-name is used as a section-name or paragraph-name, and the generated card number of statements where each procedure-name is referenced.

2.5.6.10 Interpreting Output. (Cont.)

The names appear in the order in which they appear in the source program. The number of references appearing in the cross reference dictionary for a given name is based upon the number of times the name is referenced in the code generated by the compiler. (Refer to FIGURE 2-46.)

SYMBOLS USED IN THE LISTING AND GLOSSARY TO DEFINE COMPILER-GENERATED INFORMATION

| SYMBOL | MEANING |
|---|---|
| DNM | SOURCE DATA NAME |
| SAV | SAVE AREA CELL |
| SWT | SWITCH CELL |
| TLY | TALLY CELL |
| WC | WORKING CELL |
| TS | TEMPORARY STORAGE CELL |
| VLC | VARIABLE LENGTH CELL |
| SBL | SECONDARY BASE LOCATOR |
| BL | BASE LOCATOR |
| BLL | BASE LOCATOR FOR LINKAGE SECTION |
| ON | ON COUNTER |
| PFM | PERFORM COUNTER |
| PSV | PERFORM SAVE |
| VN | VARIABLE PROCEDURE NAME |
| SBS | SUBSCRIPT ADDRESS |
| XSW | EXHIBIT SWITCH |
| XSA | EXHIBIT SAVE AREA |
| PRM | PARAMETER |
| PN | SOURCE PROCEDURE NAME |
| GN | GENERATED PROCEDURE NAME |
| DTF | DTF ADDRESS |
| VN | VARIABLE NAME INITIALIZATION |
| LIT | LITERAL |
| TS2 | TEMPORARY STORAGE (NON-ARITHMETIC) |
| RSV | REPORT SAVE AREA |
| SDF | SECONDARY DTF POINTER |
| TS3 | TEMPORARY STORAGE (SYNCHRONIZATION) |
| INX | INDEX CELL |
| V (BCDNAME) | VIRTUAL |
| VIR | VIRTUAL |

FIGURE 2-46

2.5.6.10 Interpreting Output. (Cont.)

(Circle 9) The diagnostic message associated with the compilation is always listed. The format of the diagnostic message is:

(Circle (9,A)) Compiler-generated card number is the number of a line in the source program related to the error.

(Circle (9,B)) The message identification for the Disk Operating System Full American National Standard COBOL compiler always begins with the symbols IL.A.

(Circle (9,C)) There are four severity levels as follows:

- (W) Warning. This level indicates that an error was made in the source program. However, it is not serious enough to interfere with the execution of the program. These warning messages are listed only if the FLAGW option is specified in the CBL card or chosen at system generation time.

- (C) Conditional. This level indicates that an error was made but the compiler usually makes a corrective assumption. The statement containing the error is retained. Execution can be attempted.

- (E) Error. This level indicates that a serious error was made. Usually the compiler makes no corrective assumption. The statement or operand containing the error is dropped. Compilation is completed, but execution of the program should not be attempted.

- (D) Disaster. This error indicates that a serious error was made. Compilation is not completed. Results are unpredictable.

(Circle (9,D)) The message text identifies the condition that caused the error and indicates the action taken by the compiler.

OBJECT MODULE.

- The object module contains the external symbol dictionary, the text of the program, and the relocation dictionary. It is followed by an END statement that marks the end of the module. For additional information about the external symbol dictionary and the relocation dictionary, see the publication DOS System Control and Service.

- An object deck is punched if the DECK option is specified, unless an E-level diagnostic message is generated. The object module is written on SYSLNK if the LINK option is specified, unless an E-level diagnostic message is generated.

LINKAGE EDITOR OUTPUT. The output of the link edit step may include:

- A printed listing of the job control statements.

2.5.6.10 Interpreting Output. (Cont.)

- A map of the phase after it has been processed by the Linkage Editor.
- Diagnostic messages.
- A listing of the linkage editor control statements.
- A phase which may be assigned to the core image library.

Any diagnostic messages associated with the Linkage Editor are automatically generated as output. The other forms of output may be requested by the OPTION control statement. All output to be listed is printed on the device assigned to SYSLST.

The illustration of Linkage Editor Output (FIGURE 2-50) is an example of a linkage editor output listing. It shows the job control statements and the phase map. The different types of output are numbered and each type to be explained is lettered.

Comments on the Phase Map. The severity of linkage editor diagnostic messages may affect the production of the phase map. Since various processing options affect the structure of the phase, the text of the phase map will sometimes provide additional information. For example, the phase may contain an overlay structure. In this case, a map will be listed for each segment in the overlay structure.

LINKAGE EDITOR MESSAGES. The Linkage Editor may generate informative or diagnostic messages. A complete list of these messages is included in the publication DOS Operator Communications and Messages.

DOS ANSI COBOL UNRESOLVED EXTERNAL REFERENCES.

- When the linkage editor encounters a weak external reference (WXTRN), autolinking is suppressed and the V-type address constant is either resolved from those modules included into the load module or it remains unresolved. Unresolved WXTRNs will not cause the linkage editor to cancel the link step if ACTION CANCEL is in effect.

- The object time subroutine library of the ANSI FULL COBOL compiler utilizes WXTRNs not only as address constants but also as switches to determine at object time whether certain options are in effect.

- Unresolved WXTRNs are normally intentional but unresolved WXTRNs are normally unintentional and an error.

- Any of the following unresolved WXTRNs may appear when link editing an object module produced by an ANSI COBOL compiler:

2.5.6.10 Interpreting Output. (Cont.)

```
ILBDCKP?

ILBDDSPØ

ILBDRELØ

ILBDDSP1

ILBDDSP3
```

COBOL PHASE EXECUTION OUTPUT.

- The output generated by program execution (in addition to data written on output files) may include:

  Data displayed on the console or on the printer.

  Messages to the operator.

  System informative messages.

  System diagnostic messages.

  A system dump.

- A dump and system diagnostic messages are generated automatically during program execution only if the program contains errors that cause abnormal termination.

- The following text is an explanation of an example of output from the execution job step. (Refer to FIGURE 2-47.)

2.5.6.10 **Interpreting Output.** (Cont.)

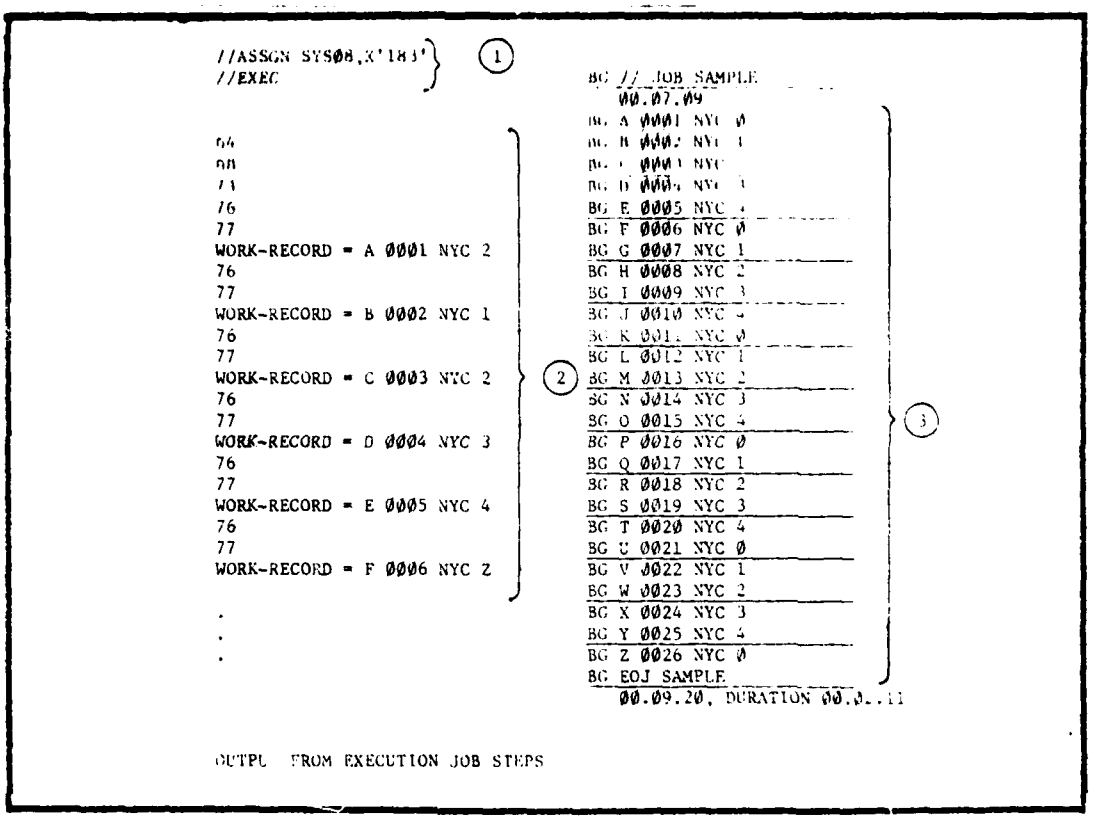



FIGURE 2-47

• Each of these messages contains an identification code in the first column of the message to indicate the portion of the operating system that generated the message. The following table lists these codes, together with identification for each. (Refer to FIGURE 2-48.)

| CODE | IDENTIFICATION |
|---|---|
| 0 | An on-line console message from the Supervisor |
| 1 | A message from the Job Control Processor |
| 2 | A message from the Linkage Editor |
| 3 | A message from the Librarian |
| 4 | A message from LIOCS |
| 7 | A message from the Sort program |
| C | A message from COBOL object-time subroutines |

FIGURE 2-48

2.5.6.10 Interpreting Output. (Cont.)

LINKAGE EDITOR OUTPUT.

- (Circle 1) The job control statements are listed since the LOG option is specified.

- (Circle 2) Disk linkage editor diagnostic message of input. The ACTION statement is not required. If the MAP option is specified, SYSLST must be assigned. If the statement is not used and SYSLST is assigned, MAP is assumed and a map of main storage and any error diagnostic messages are considered output on SYSLST.

- (Circle 3) Map of main storage. A phase map is printed when MAP is specified (or assumed) during linkage editor processing. The following information is contained in the map of main storage:

- (Column A) The name of each phase. This is the name specified in the phase statement.

- (Column B) The transfer address of each phase.

- (Column C) The lowest main storage location of each phase.

- (Column D) The highest main storage location of each phase.

- (Column E) The hexadecimal disk address where the phase begins in the core image library.

- (Column F) The names of all CSECT's belonging to a phase.

- (Column G) All defined entry points within a CSECT. If an entry point is not referenced, it is flagged with an asterisk (*).

- (Column H) The address where each CSECT is loaded.

- (Column J) The relocation factor of each CSECT.

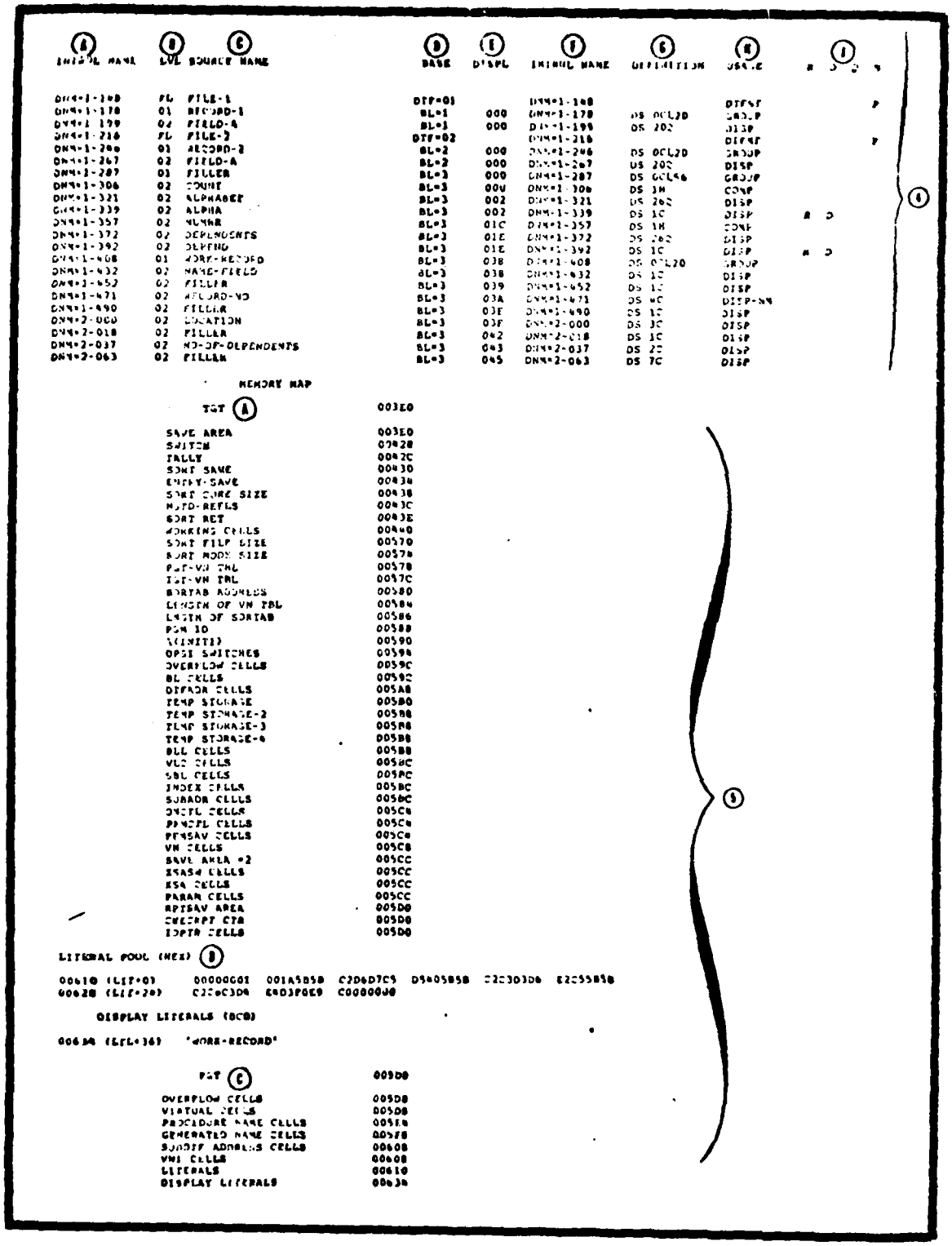
2.5.6.10 Interpreting Output. (Cont.)

ILLUSTRATION OF COMPILER OUTPUT

```
// JOB SAMPLE
// OPTION NODECK,LINK,LIST,:ISTX,SYM,ERRS                    (1)
// EXEC FCOROL

 CAL QUOTE,SEQ                                              (2)
00001    000010 IDENTIFICATION DIVISION.
00002    000020 PROGRAM-ID. TESTRUN.
00003    000030     AUTHOR. PROGRAMMER NAME.
00004    000040     INSTALLATION. NEW YORK PROGRAMMING CENTER.
00005    000050     DATE-WRITTEN.  FEBRUARY 2,1971
00006    000060 DATE-COMPILED. 04/24/71
00007    000070    REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008    000080     COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
00009    000090     INPUT.
00010    000100
00011    000110 ENVIRONMENT DIVISION.
00012    000120 CONFIGURATION SECTION.
00013    000130 SOURCE-COMPUTER. IBM-360-H50.
00014    000140 OBJECT-COMPUTER. IBM-360-H50.
00015    000150 INPUT-OUTPUT SECTION.
00016    000160 FILE-CONTROL.
00017    000170     SELECT FILE-1 ASSIGN TO SYS008-UT-2400-S.
00018    000180     SELECT FILE-2 ASSIGN TO SYS008-UT-2400-S.
00019    000190                                                (3)
  .
  .
  .
00056    000550 PROCEDURE DIVISION.
00057    000560 BEGIN. READY TRACE.
00058    000570     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00059    000580     AND INITIALIZES COUNTERS.
00060    000590 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO COUNT, NUMBR.
  .
  .
  .
00073    000720 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00074    000730     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES
00075    000740     OUT EMPLOYEES WITH NO DEPENDENTS.
00076    000750 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00077    000760 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00078    000770     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO STEP-6.
00079    000780 STEP-8. CLOSE FILE-2.
00080    000790     STOP RUN.
```

FIGURE 2-49

2.5.6.10 Interpreting Output. (Cont.)



```
(A) INTERNAL NAME   (B) LVL (C) SOURCE NAME          (D) BASE  (E) DISPL  (F) INTERNAL NAME   (G) DEFINITION   (H) USAGE   (I) A S O V

DNM=1-148      FL  FILE-1                    DTF=01              DNM=1-148                    DTFSF          P
DNM=1-178      01  RECORD-1                  BL=1    000         DNM=1-178      DS  0CL20     GROUP
DNM=1-195      02  FIELD-A                   BL=1    000         DNM=1-195      DS  20C       DISP
DNM=1-216      FL  FILE-2                    DTF=02              DNM=1-216                    DTFSF          P
DNM=1-246      01  RECORD-2                  BL=2    000         DNM=1-246      DS  0CL20     GROUP
DNM=1-267      02  FIELD-A                   BL=2    000         DNM=1-267      DS  20C       DISP
DNM=1-287      01  FILLER                    BL=3    000         DNM=1-287      DS  6CL46     GROUP
DNM=1-306      02  COUNT                     BL=3    00U         DNM=1-306      DS  1H        COMP
DNM=1-321      02  ALPHABET                  BL=3    002         DNM=1-321      DS  26C       DISP
DNM=1-339      02  ALPHA                     BL=3    002         DNM=1-339      DS  1C        DISP     A  O
DNM=1-357      02  NUMBR                     BL=3    01C         DNM=1-357      DS  1H        COMP
DNM=1-372      02  DEPENDENTS                BL=3    01E         DNM=1-372      DS  2C        DISP
DNM=1-392      02  DEPEND                    BL=3    01E         DNM=1-392      DS  1C        DISP     A  O
DNM=1-408      01  WORK-RECORD               BL=3    038         DNM=1-408      DS  0CL20     GROUP
DNM=1-432      02  NAME-FIELD                BL=3    038         DNM=1-432      DS  1C        DISP
DNM=1-452      02  FILLER                    BL=3    039         DNM=1-452      DS  1C        DISP
DNM=1-471      02  RECORD-NO                 BL=3    03A         DNM=1-471      DS  4C        DISP-NM
DNM=1-490      02  FILLER                    BL=3    03E         DNM=1-490      DS  1C        DISP
DNM=2-000      02  LOCATION                  BL=3    03F         DNM=2-000      DS  3C        DISP
DNM=2-018      02  FILLER                    BL=3    042         DNM=2-018      DS  1C        DISP
DNM=2-037      02  NO-OF-DEPENDENTS          BL=3    043         DNM=2-037      DS  2C        DISP
DNM=2-063      02  FILLER                    BL=3    045         DNM=2-063      DS  7C        DISP

                        MEMORY MAP

                    TGT (A)                 003E0

                SAVE AREA                   003E0
                SWITCH                      003E8
                TALLY                       003EC
                SORT SAVE                   003F0
                ENTRY-SAVE                  003F4
                SORT CORE SIZE              003F8
                MOD-REFLS                   003FC
                SORT RET                    003FE
                WORKING CELLS               00400
                SORT FILE SIZE              00570
                SORT MODE SIZE              00578
                FCT-VN TBL                  00578
                TCT-VN TBL                  0057C
                SORTAB ADDRESS              00580
                LENGTH OF VN TBL            00584
                LNGTH OF SORTAB             00586
                PGM ID                      00588
                XXXINIT3                    00590
                OPSI SWITCHES               00594
                OVERFLOW CELLS              0059C
                BL CELLS                    0059C
                DIFADR CELLS                005A8
                TEMP STORAGE                005B0
                TEMP STORAGE-2              005B8
                TEMP STORAGE-3              005B8
                TEMP STORAGE-4              005B8
                BLL CELLS                   005B8
                VLS CELLS                   005BC
                SBL CELLS                   005BC
                INDEX CELLS                 005BC
                SUBADR CELLS                005BC
                DNCTL CELLS                 005C4
                PRNCPL CELLS                005C4
                PRNSAV CELLS                005C4
                VN CELLS                    005C8
                SAVE AREA #2                005CC
                ESAS4 CELLS                 005CC
                ESA CELLS                   005CC
                PARAM CELLS                 005CC
                RPTSAV AREA                 005D0
                OVERPT CTR                  005D0
                IDPTR CELLS                 005D0

LITERAL POOL (HEX) (D)

006I0 (LIT+0)     00000G01  001A5B5B  C3D6D7C5  D5405B5B  E2C3D3D6  E2C55B5B
00620 (LIT+20)    C3C9C3D9  E4D3FGE9  C0000J40

     DISPLAY LITERALS (DC'0)

00634 (LIT+36)   'WORK-RECORD'

                    PGT (C)                 009D0

                OVERFLOW CELLS              009D8
                VIRTUAL CELLS               009D8
                PROCEDURE NAME CELLS        009E8
                GENERATED NAME CELLS        009F8
                SUBDIF ADDRESS CELLS        00608
                VNI CELLS                   00608
                LITERALS                    00610
                DISPLAY LITERALS            00634
```

FIGURE 2-49 (Cont.)

2.5.6.10 Interpreting Output. (Cont.)

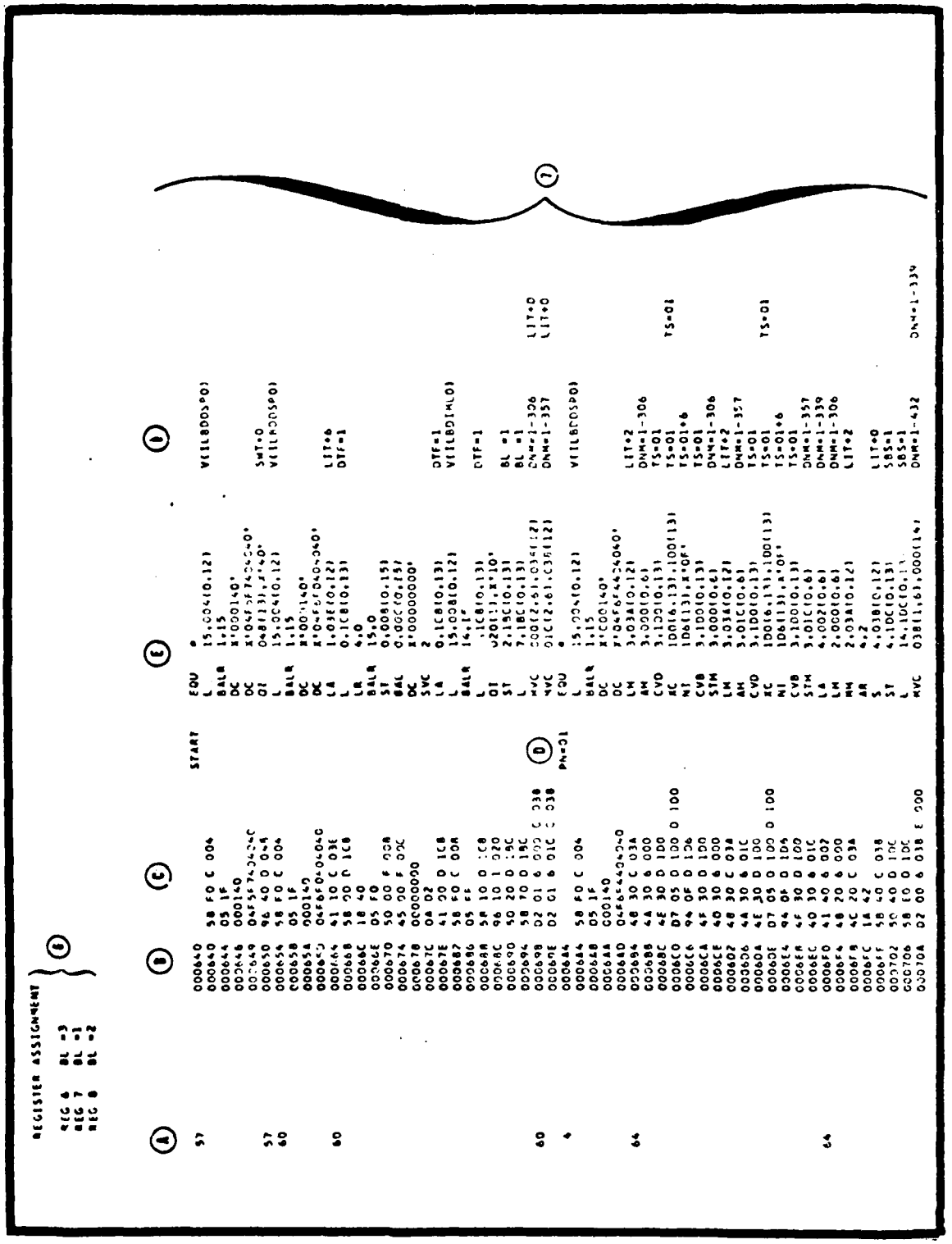



FIGURE 2-49 (Cont.)

2.5.6.10 Interpreting Output. (Cont.)

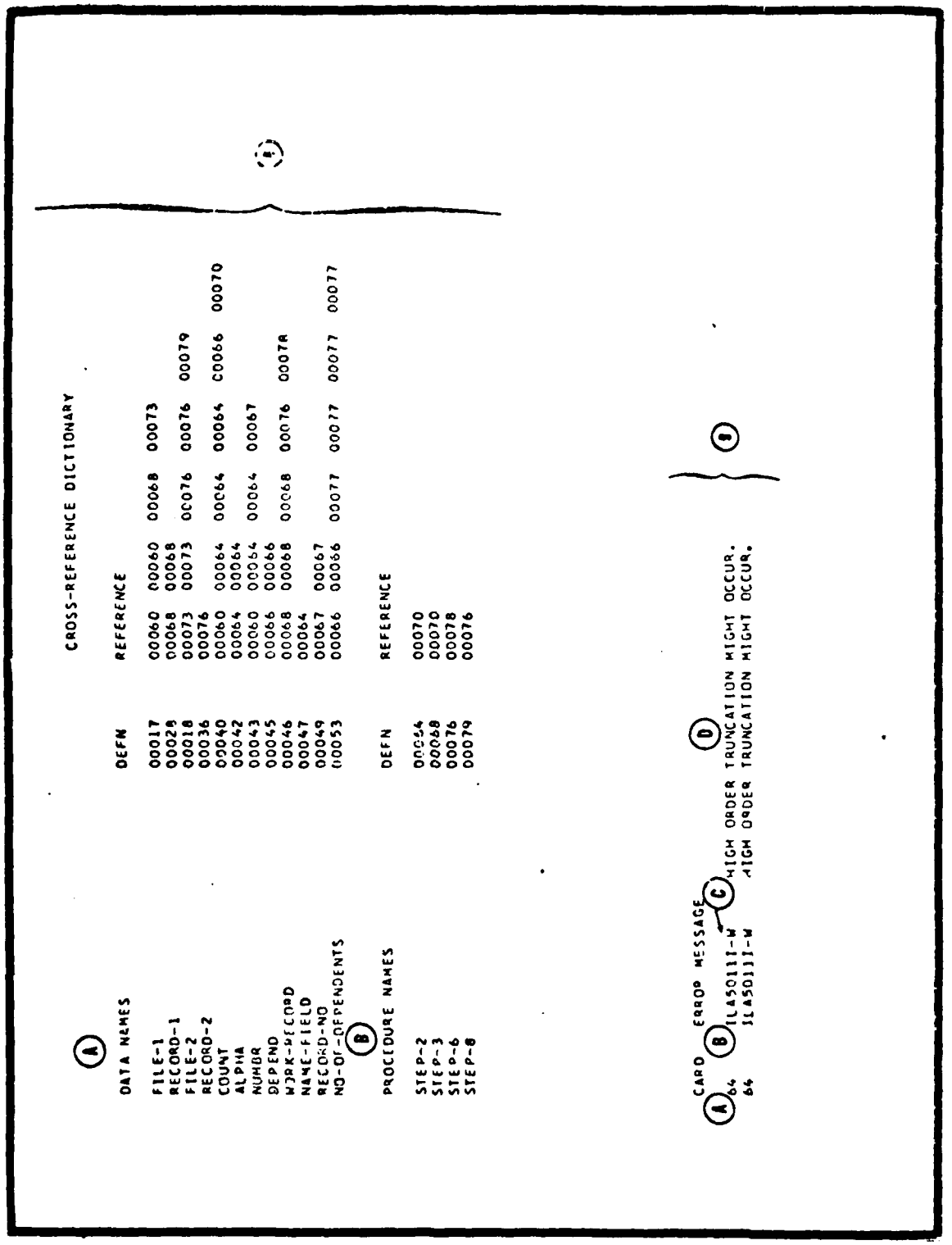



FIGURE 2-49 (Cont.)

2.5.6.10 Interpreting Output. (Cont.)

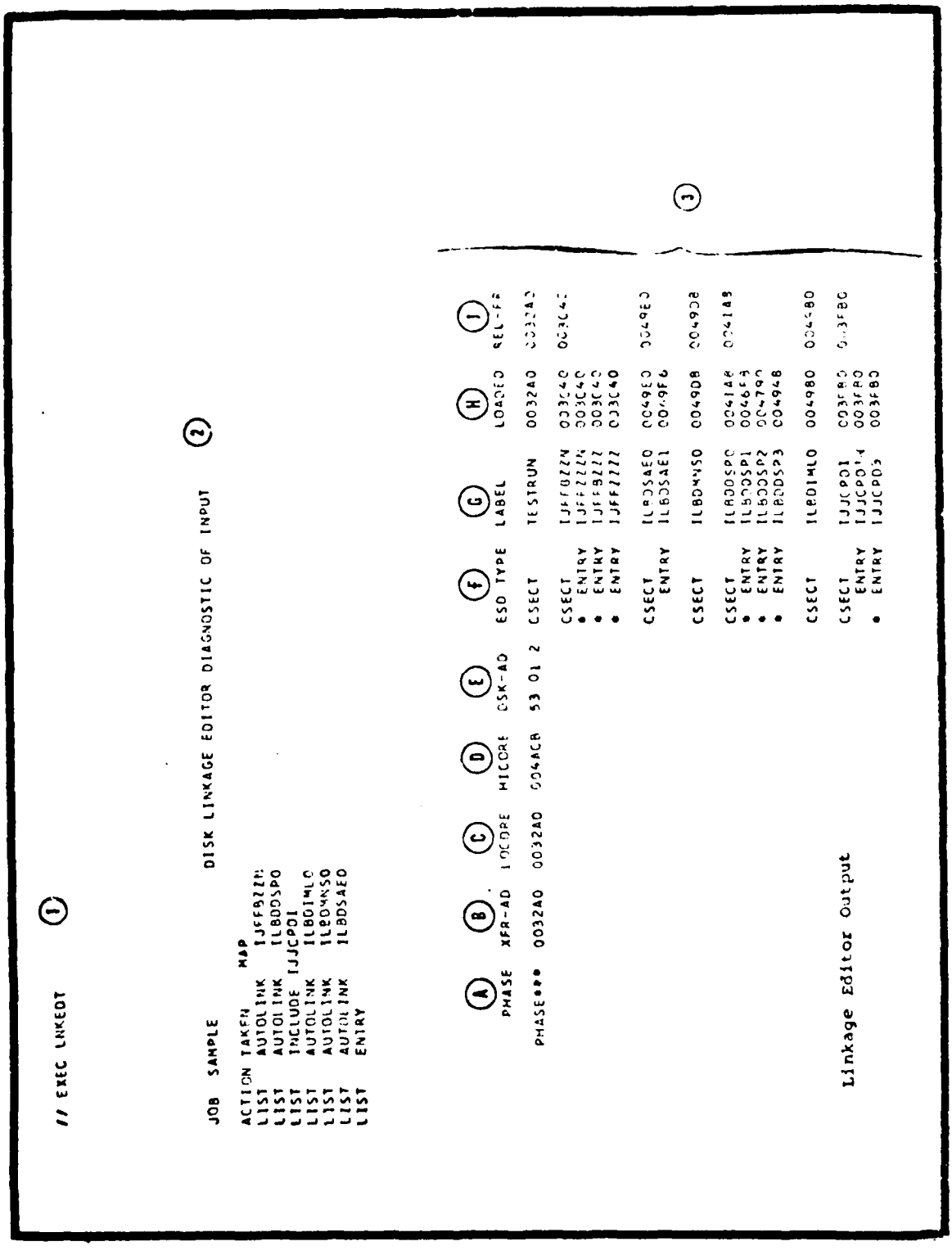



FIGURE 2-50

2.5.6.11 OS COBOL Program Debugging Aids.

INTRODUCTION.

• As the result of testing COBOL programs, abnormal dumps may occur. The reason for abnormal termination is usually indicated by a system completion code which appears on the ABEND dump listing. A SYSUDUMP or SYSABEND JCL card must be provided for each step where a core dump is required.

• The general explanation for system completion codes is usually in the IBM Messages and Codes Manual. For some codes, the explanation may not be adequate for determining the exact reason for the abnormal termination of the program. For additional explanation of COBOL core dumps, see Programers Guide to Debugging.

• Following is a list of the most commonly encountered system completion codes. For each one, the error is described, possible causes listed and additional reference information given. Not all of the information provided will apply in every case, but, as far as possible, common causes and sources of information are pointed out. The completion codes are listed in sequence for easy reference.

2.5.6.12 COMPLETION CODE - ØØ1.

ERROR DESCRIPTION.

• Input/Output error, issued under QSAM or BSAM.

• No SYNAD exit was specified.

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.

ADDITIONAL REFERENCE INFORMATION.

• Before proceeding, refer to Messages and Codes on listing number IECØ2Ø1.

2.5.6.12 COMPLETION CODE - ØØ1. (Cont.)

- Register 1 of the SVRB for SVC 55 (end-of-volume) points to DCB for which the I/O error was found.

- Register 2 also points to the DCB (if these registers vary, register 2 is more likely to be correct; register 1 is altered regularly by several supervisor routines).

- Register Ø points to the IOB-8.

- Address of the DCB + hex 2D (dec. 45): DEB address.

- Address of the DEB + hex 21 (dec. 33): UCB address.

- Address of the DCB + hex 44 (dec. 68): IOB prefix address.

- First IOB prefix + 1: Second IOB prefix address.

- Second IOB prefix + 1: third IOB prefix address, etc.

- Register 4 contains a pointer to the current IOB prefix.

- Register 6 should contain the same pointer as register 4.

- IOB prefix address + 8: Standard IOB fields.

- First byte of IOB, IOBFLAG1 field, bit 5 on indicates permanent error.

- IOB address + 2: first and second sense bytes.

- IOB address + 4: event control block (ECB) completion code.

- IOB address + 5: address of ECB.

- IOB address + hex C (dec. 12): two bytes, status information of the last channel status word (CSW).

- Register 5 contains the first byte of flags (IOBFLAG1) and the first and second sense bytes.

- IOB address + hex 11 (dec. 17): address of channel program.

- IOB address + hex 2Ø (dec. 32): or hex 28 (dec. 4Ø) for direct access devices: channel program beginning.

2.5.6.13 COMPLETION CODE - Ø13.

ERROR DESCRIPTION.

- Detection of conflicting or unsupported parameters during OPEN processing.

PROBABLE CAUSES.

- Member name specified in a DD statement could not be found.
- A directory allocation subparameter was not specified in a DD statement for a partitioned data set.
- Conflicting or incomplete DCB subparameters.
- No BLKSIZE DCB subparameter specified for a DUMMY data set.
- Default SYSIN or SYSOUT blocking (specified in OS procedures) conflicts with problem program specifications.
- Track overflow or updating may be attempted for an operating system version that does not support these features.

ADDITIONAL REFERENCE INFORMATION.

- Before proceeding, refer to Messages and Codes message number IEC 141I.
- The PSW field of the dump (not the APSW field) gives the next instruction to be executed in the problem programs.
- The loader or linkage editor map can help specify if this instruction is in the mainline processing or in a called module.

2.5.6.14 COMPLETION CODE - Ø31.

ERROR DESCRIPTION.

- An input/output error was detected when processing under the indexed sequential access method (QISAM).

PROBABLE CAUSES.

- Physical damage to recording medium or device.
- Out of sequence key when loading an ISAM data set.
- Wrong length record or block.

2.5.6.14 COMPLETION CODE - Ø31. (Cont.)

ADDITIONAL REFERENCE INFORMATION.

• The DCB for the file which contains the I/O error can be found as follows:

Note address of 'INTERRUPT'.

Find the module in which the interrupt occurred in the 'LOAD LIST'.

Under 'SAVE AREA TRACE' find module name that was ENTERED VIA CALL. The return address in the problem program is listed under 'RET'. The last macro instruction (calling an SVC instruction) in the problem program before this address was the one executing when the ABEND occurred.

The DCBEXCD1 and DCBEXCD2 fields of the ISAM DCB indicate the cause of the error. DCBEXCD1 is at offset, hex 5Ø (dec. 8Ø) of the DCB access method interface for ISAM section, DCBEXCD2 is at offset 51 (dec. 81). These fields contain:

DCBEXCD1

| Bit | Meaning |
|---|---|
| Ø | Lower limit key not found |
| 1 | Invalid device address for lower limit |
| 2 | Space not found |
| 3 | Invalid request |
| 4 | Incorrectable input error |
| 5 | Incorrectable output error |
| 6 | Unreachable block |
| 7 | Overflow record |

2.5.6.14 COMPLETION CODE - Ø31. (Cont.)

DCBEXCD:

| Bit | Meaning |
|---|---|
| Ø | Sequence check |
| 1 | Duplicate record |
| 2 | DCB closed when error detected |
| 3 | Overflow record |
| 4-7 | Reserved bits |

A SYNAD routine may be helpful in pinpointing the cause of error.

2.5.6.15 COMPLETION CODE - Ø3B.

ERROR DESCRIPTION.

- The error occurred during OPEN processing for an indexed sequential data set.

PROBABLE CAUSES.

- The data set had not been created.
- The data control block had not been closed after the data set had been created.

ADDITIONAL REFERENCE INFORMATION.

- An indexed sequential file must be created and closed before it can be accessed.
- Be certain that the DCB subparameter MACRF specifies an indexed sequential data set (Assembly Language).
- Be certain that indexed file creation and retrieval is specified in the source program code (COBOL).
- The presence of a FORMAT 2 data set control block (DSCB) on a direct access device indicates the creation of an indexed sequential data set.

2.5.6.16 COMPLETION CODE - Ø3D.

ERROR DESCRIPTION.

- The error occurred during OPEN processing for an indexed sequential or direct data set.

PROBABLE CAUSES.

- Indexed sequential organization not specified in the DSORG subparameter of the DCB (this is required, even if indexed organization is specified in the source program).

- Not all volume serial numbers were specified, or they were not in correct sequence.

ADDITIONAL REFERENCE INFORMATION.

- Before proceeding, refer to Messages and Codes message number IEC 1565.

- For indexed sequential, the volume containing the index must be described in the first job control language statement.

- The number of volumes and the number of units to which these volumes will be mounted must be the same; all volumes must be mounted.

2.5.6.17 OCx COMPLETION CODE NOTE.

- A number of similarities exist among the debugging techniques indicated for the OC1, OC2, OC4, OC5, OC6, and OC7 completion codes that follow. These techniques are also applicable to other OCx completion codes not reviewed in this analysis, namely OC3, OC8, OC9, OCA, OCB, OCC, OCD, OCE and OCF. These latter codes are rarely encountered.

- Because of the similarity of the processing leading to these codes, check other OCx dump reference pages if the problems cannot be found among the reasons given for your particular OCx dump. Many of these techniques pertain to all OCx completion codes.

2.5.6.18 COMPLETION CODE - OC1.

ERROR DESCRIPTION.

- The operation code detected is not valid or has not been implemented on this model S/36Ø or S/37Ø.

PROBABLE CAUSES.

- A branch to a data area; fetching of data as an operation code for this instruction.

2.5.6.18 COMPLETION CODE - OC1. (Cont.)

- A missing or misspelled DD statement.

- A data set had not been opened when an input/output instruction was issued for it.

- A data set had been closed when an input/output instruction was issued for it (this may also cause an OC5 termination).

ADDITIONAL REFERENCE INFORMATION.

- Check register 1 or 2 at entry to ABEND + hex 28 (dec. 40).

- This should point to the DDNAME of the DD statement in error. Register 2 will be the best indicator if the addresses differ.

- The APSW field has the address of the next instruction to be executed in the problem program.

2.5.6.19 COMPLETION CODE - OC5.

ERROR DESCRIPTION.

- An address is specified that is outside of the available storage of the particular computing system.

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 the DCB.

- A missing or misspelled DD statement.

- An attempt to CLOSE a data set 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.

2.5.6.19 COMPLETION CODE - 0C5. (Cont.)

• COBOL: an attempt to reference an input/output area before a READ or OPEN statement has been issued for the file.

ADDITIONAL REFERENCE INFORMATION.

• Register 1 at entry to ABEND contains the address of the DCB.

• The address plus hex 28 (dec. 40) contains the DDNAME of the data set involved.

• Register 14 points to the instruction following the input/output instruction.

• The APSW field contains the address of the next instruction to be executed in the problem program.

2.5.6.20 COMPLETION CODE - 0C7.

ERROR DESCRIPTION.

• Data in a field was of incorrect format for the instruction attempting to process it.

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 (0C5) or protection (0C4) 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 (0C5) or protection (0C4) completion code.

• ASSEMBLY LANGUAGE: The sign or digit codes of operands in decimal arithmetic, editing operations or the CONVERT TO BINARY (CVB) instruction are incorrect.

2.5.6.20 COMPLETION CODE - 0C7. (Cont.)

- COBOL: Data was moved from the DISPLAY field to the COMPUTATIONAL field at group level.

- COBOL: The figurative constants ZERO or LOW-VALUE were moved to a group level numeric field.

- COBOL: Omission of USAGE clause or inclusion of an erroneous USAGE clause.

- Incorrect linkage section data definition, passing parameters in the wrong order, omission or inclusion of a parameter, failure to carry over a USAGE clause when necessary, defining the length of a parameter incorrectly.

ADDITIONAL REFERENCE INFORMATION.

- Registers 1 or 2 point to the DCB for the last referenced file.

- Register 9 may contain the address of the DDNAME of the last referenced file.

- Register 8 and/or register 10 may contain the UCB address for the last referenced file.

- Register 4 + hex 64 (dec. 100) may contain the DSNAME of the last referenced file.

- The APSW field contains the address of the next instruction to be executed in the problem program.

2.5.6.21 COMPLETION CODE - 237.

ERROR DESCRIPTION.

- The end-of-volume routine (SVC 55) detected an error at the end-of-volume or while positioning the second or subsequent volume for processing.

PROBABLE CAUSES.

- A verification error occurred in label processing (an incorrect label may have been encountered).

- The tape label block count did not agree with the DCB block count (probably because of a skipped block due to hardware difficulties).

2.5.6.21 COMPLETION CODE - 237. (Cont.)

- The data set name specified in the DSN parameter of the DD statement was not the same as the data set name in header label 1 of a tape volume.
- An incorrect volume was mounted.
- The volume serial number was specified incorrectly in the SER sub-parameter of the DD statement.

ADDITIONAL REFERENCE INFORMATION.

- Before proceeding, refer to Messages and Code message number IEC 023I.
- For "incorrect volume" serial number: Register 2 contains the address of the DCB.
- Address of DCB + hex 28 (dec. 40): the address of a 2-byte field giving the TIOT offset.
- The TIOT address + TIOT offset is the address of the DDNAME entry in the TIOT.
- The DDNAME entry + 4: 8-byte DDNAME in the TIOT.
- Register 9 may contain the 8-byte DDNAME address.
- Register 8 may contain the UCB address.
- Address of the UCB + hex 1C (dec. 28): volume serial number of the volume presently mounted.
- Register 4 contains the address of the label in storage.
- Register 4 + hex 64 (dec. 100): DSN in storage.
- (SOURCE: DD statement VOL=SER=parameter). Up to the first five volume serial numbers are adjacently arranged here.
- Register 5 points to the DEB.

FOR BLOCK COUNT DISCREPANCY.

- Register 2 contains the address of the DCB.
- Address of the DCB + hex 28 (dec. 40): The address of a 2-byte field giving the TIOT offset.
- The TIOT address + the TIOT offset is the address of the DDNAME entry in the TIOT.

2.5.6.21 COMPLETION CODE - 237. (Cont.)

- The DDNAME entry + 4: 8-byte DDNAME in the TIOT.
- Register 9 may contain the 8-byte DDNAME address.
- The address of the DDNAME + hex 1Ø (dec. 16): UCB address (one word).
- Register 1Ø may contain the UCB address.
- Address of the UCB + hex 1C (dec. 28): volume serial number of the volume presently mounted.
- Register 4 contains the address of the label in storage.
- Register 4 + hex 64 (dec. 1ØØ): DSN in storage (SOURCE: DD statement or catalog).
- Register 4 + hex DA (dec. 218): volume serial number (SOURCE: DD statement or VOL=SER=parameter).
- Up to the first five volume serial numbers are arranged in order.
- Register 5 points to the DEB.
- The address of the DCB + hex C (dec. 12): block count field of the DCB.
- If register 4 + hex 36 (dec. 54): block count in label (decimal notation).
- Register 12 contains the block count also (hexadecimal notation).

2.5.6.22 COMPLETION CODE - 637.

ERROR DESCRIPTION.

- An incorrectable I/O error occurred during end-of-volume processing on a tape just read or a new volume just mounted.
- Concatenated data sets have unlike attributes.

PROBABLE CAUSES.

- An I/O error was encountered in writing a tape mark.
- An I/O error was encountered in positioning the tape.

2.5.6.22 COMPLETION CODE - 637. (Cont.)

- An I/O error was encountered in reading a label.
- An I/O error was encountered in sensing for a file protection ring.

ADDITIONAL REFERENCE INFORMATION.

- Before proceeding, refer to Messages and Codes message number IEC Ø26I.
- I/O errors frequently indicate a hardware malfunction.
- If data sets with unlike attributes (e.g., blocksize) are being concatenated, the DCBOFLGS in the DCB must be set to indicate a concatenation of unlike attributes.
- Register 2 contains the address of the DCB being OPENed.
- Address of the DCB + hex 28 (dec. 4Ø): the address of the 2-byte field containing the TIOT offset.
- Register 9 may contain the address of the DDNAME field in the TIOT.
- TIOT address + TIOT offset: address of the DDNAME entry in the TIOT.
- The DDNAME entry + 4: 8-byte DDNAME in the TIOT.
- Register 4 points to the label read into storage.
- Register 4 + hex 64 (dec. 1ØØ) contain the DSNAME (SOURCE: DD statement).
- Register 4 + hex 11Ø (dec. 272): IOB prefix for OPEN.
- Address of IOB + 8 (for BSAM, QSAM, BPAM): IOB standard fields.
- The first byte of the IOB contains the IOBFLAG1 byte: If bit 5 is on, a permanent error has been encountered.
- The IOB + 2 contains the first and second sense bytes.
- The IO3 + 4 contains the ECB completion code.
- The IOB + hex C (dec. 12) contains the two status bytes of the channel status word (CSW).

2.5.6.22 COMPLETION CODE - 637. (Cont.)

- The IOB + hex 11 (dec. 17) contains the starting address of the channel program.

- Register 1Ø contains the address of the UCB.

2.5.6.23 COMPLETION CODE - 8Ø4.

ERROR DESCRIPTION.

- More storage was requested than was currently available in the region.

PROBABLE CAUSES.

- The REGION parameter in JOB or EXEC statement does not specify enough storage for the processing program. (If a REGION is specified on the JOB card, REGION parameters on EXEC cards are ignored.)

- If a REGION parameter was not included, the default region size for the installation was too small.

- Blocking factors were increased without correspondingly increasing the REGION size request to accommodate the larger physical blocks.

- A cataloged procedure step requested more storage than was available.

ADDITIONAL REFERENCE INFORMATION.

- Be sure the SIZE parameter, sometimes used to pass storage size information to the processing program (such as compilers, sort/merge, etc.) be compatible with the storage provided in the region.

- ASSEMBLY LANGUAGE: In a dynamic programing environment, be certain that FREEMAIN requests are regularly issued before GETMAIN requests.

- Check to be certain that a REGION parameter wasn't incorrectly overridden.

- Completion code 8ØA is closely related to this completion code.

2.5.6.24 COMPLETION CODE - 8Ø6.

ERROR DESCRIPTION.

- A requested program module could not be found.

PROBABLE CAUSES.

- A JOBLIB or STEPLIB DD statement was missing (a common error).

2.5.6.24 COMPLETION CODE - 806. (Cont.)

- The module name was misspelled.
- The library data set had been deleted from the device.
- An unrecoverable input/output error occurred while searching the directory in order to retrieve the program.

ADDITIONAL REFERENCE INFORMATION.

- Register 15 contains 00000004 if the requested module could not be found in the private library, job library or link library (SYS1.LINKLIB).
- Register 15 contains 00000008 if an uncorrectable I/O occurred in searching the library directory.
- Register 12 + 4 bytes is the location of the 8-byte name of the requested program that could not be loaded.

2.5.6.25 COMPLETION CODE - 813.

ERROR DESCRIPTION.

- The error occurred during OPEN processing because the data set name on a magnetic tape volume did not match the DSNAME specified for it through job control language.

PROBABLE CAUSES.

- The volume serial number specified through job control language or through the catalog was incorrect.
- The DSNAME parameter is misspelled.
- The wrong volume is mounted.

ADDITIONAL REFERENCE INFORMATION.

- Before proceeding, refer to Messages and Codes message number IEC 149I.
- Register 2 contains the address of the DCB.
- The DCB address + hex 28 (dec. 40): the address of a 2-byte field containing the TIOT offset.
- The address in the TCBTIO field (word 4 of the TCB) + the TIOT offset: address of DDNAME entry in TIOT.

2.5.6.25 COMPLETION CODE - 813. (Cont.)

- The TIOT address may be contained in register 9.
- Register 14 may contain the address of the DDNAME entry in the TIOT.
- The DDNAME entry + 4 is the start of the 8-byte DDNAME field in the TIOT.
- Register 4 has the address of the label in storage.
- Register 4 + 4 is the low-order 17 bytes of the tape label as represented on the tape label (this follows the HDRI identification).
- Register 4 + 64 contains the DSNAME in storage (SOURCE: DD statement).
- Register 11 may contain this DSNAME address.
- Register 4 + hex 11Ø (dec. 272): IOB prefix for OPEN.
- IOB address + 8 contains the IOB standard fields (BSAM, QSAM, BPAM).
- The first byte of the IOB contains the IOBFLAG1 byte. If bit 5 is on, a permanent error is indicated.
- The IOB + 2 contains the first and second sense bytes.
- The IOB + 4 contains the ECB completion code (if the high order bit is not on, an error condition occurred).
- The IOB + hex C (dec. 12) contains the two bytes of CSW status flag information.
- The IOB + hex 11 (dec. 27) contains a pointer to the start of the channel program.
- Register 1Ø may contain the UCB address.

2.5.6.26 COMPLETION CODE - D37.

ERROR DESCRIPTION.

- Space allocated for a data set on a direct access device was exceeded. No secondary allocation was specified. The error was detected by end-of-volume processing.

PROBABLE CAUSES.

- The SPACE parameter did not request any secondary storage area for the data set.

2.5.6.26 COMPLETION CODE - D37. (Cont.)

• A related completion code, B37, is encountered if secondary space was specified and all primary and secondary extents were exceeded.

ADDITIONAL REFERENCE INFORMATION.

• Before proceeding, refer to Messages and Codes message IEC Ø31I for D37 or IEC Ø3Ø1 for B37.

• The DEB indicates the physical location and size of the single extent created for this data set.

• Register 2 contains the address of the DCB.

• DCB address + hex 28 (dec. 4Ø): address of the 2-byte TIOT offset field in the DCB.

• TIOT address (from 4th word in TCB) + the TIOT offset is the address of the DDNAME field in the TIOT.

• Register 9 may also contain the DDNAME field address.

• DDNAME field in TIOT + 4 gives the starting address of the 8-byte DDNAME stored in the TIOT.

• Register 4 + hex 64 (dec. 1ØØ): DSNAME (SOURCE: DD statement).

• Register 1Ø contains the address of the UCB.

2.5.6.27 COMPLETION CODE - E37.

ERROR DESCRIPTION.

• The usual error is that space is exceeded when writing a partitioned data set on a direct access device. The error is detected by end-of-volume processing routines.

PROBABLE CAUSES.

• Sixteen extents had been written for a partitioned data set on a direct access device, but additional space was needed.

• For a partitioned data set creating extents beyond the first, additional space was needed, but no more space was available on the volume (a partitioned data set cannot be continued on a second volume).

• For tape or direct access, not enough volumes were specified but more space was needed.

2.5.6.27 COMPLETION CODE - E37. (Cont.)

ADDITIONAL REFERENCE INFORMATION.

- Before proceeding, refer to Messages and Codes message IEC Ø321.
- The DEB contains the physical starting and ending address of the data set on the device.
- Register 2 contains the address of the DCB.
- The DCB address + hex 28 (dec. 4Ø) contains the 2-byte TIOT offset field.
- The TIOT address (from word 4 of the TCB) + the TIOT offset references the DDNAME entry in the TIOT.
- Register 9 may contain the DDNAME field address.
- DDNAME field in the TIOT + 4: 8-byte DDNAME entry in the TIOT.
- Register 4 contains the address of the volume serial number.
- Register 4 + hex 64 (dec. 1ØØ): DSNAME (SOURCE: DD statement).
- Register 4 + hex DA (dec. 218): address of volume serial number (SOURCE: DD statement or system catalog).
- Register 1Ø contains the address of the UCB.
- Register 12 may contain the address of the volume serial number in storage.

2.5.6.28 Control Block Pointers.

- This paragraph summarizes the contents of the control blocks that are most useful in debugging. Control blocks are presented in alphabetical order, with displacements in decimal, followed by the hexadecimal counterpart in parentheses.

CVT - Communications Vector Table.

| | |
|---|---|
| +Ø | Address of TCB control words |
| +53(35) | Address of entry point of ABTERM |
| +193(C1) | Address of secondary CVT (used only with Model 65 Multiprocessing systems and TSO) |

2.5.6.28. Control Block Pointers. (Cont.)

DCB - Data Control Block.

| | |
|---|---|
| +4Ø(28) | ddname (before open); offset to ddname in TIOT (after open) |
| +45(2D) | DEB address |
| +48(30) | TCB address |

DEB - Data Extent Block.

| | |
|---|---|
| +1 | TCB address |
| +5 | Address of next DEB |
| +25(19) | DCB address |
| +33(21) | UCB address |
| +38(26) | Address of start of extent |
| +42(2A) | Address of end of extent |

ECB - Event Control Block.

| | |
|---|---|
| +1 | RB address or completion code |

2.5.6.28 Control Block Pointers. (Cont.)

IOB - Input/Output Block.

| | |
|---|---|
| -7 | Address of next IOB (BSAM, QSAM, and BPAM) |
| +2 | Sense bytes |
| +5 | ECB address |
| +9 | CSW |
| +17(11) | CCW list address |
| +21(15) | DCB address |

RB - Request Block (PCP and MFT).

| | |
|---|---|
| -8 | Address of previous RB on load list |
| -4 | Address of next RB on load list |
| +Ø | Module name |
| +13(D) | Entry point address |
| +16(1Ø) | Resume PSW |
| +29(1D) | Address of previous RB |

RB - Request Block (MVT).

| | |
|---|---|
| +4 | Last half of user's PSW |
| +13(D) | CDE address |
| +16(1Ø) | Resume PSW |
| +29(1D) | Address of previous RB |

2.5.6.28 Control Block Pointers. (Cont.)

TIOT - Task Input/Output Table.

| | |
|---|---|
| +Ø | Job name |
| +8 | Step name |
| +24(18) | DD entries begin (one variable-length entry for each DD Statement) |
| +Ø | Length of DD entry |
| +4 | ddname |
| +16(1Ø) | Device entries begin (one 4-byte entry for each device) |
| +2Ø(14) | Next device entry (if there is one) |

.
.
.

(Next DD entry begins at 24 (18) plus length of first DD entry)

TCB - Task Control Block (PCP and MFT).

| | |
|---|---|
| +1 | Address of most recent RB |
| +9 | Address of most recent DEB |
| +13(D) | TIOT address |
| +16(1Ø) | Completion code |
| +25(19) | MSS boundary box address |
| +37(25) | Address of most recent RB on load list |
| +113(71) | Address of first save area |
| +161(A1) | Address of STAE control block |
| +181(B5) | Address of the job step control block |

2.5.6.28 Control Block Pointers. (Cont.)

TCB - Task Control Block.

(MFT) With Subtasking.

| | |
|---|---|
| +45(2D) | Address of TCB for job step task |
| +129(81) | Address of TCB for next subtask attached by same parent task |
| +133(85) | Address of TCB for parent task |
| +137(89) | Address of TCB for most recent subtask |
| +141(91) | Address of ECB to be posted at task completion |
| +181(B5) | Address of the job step control block |

TCB - Task Control Block (MVT).

| | |
|---|---|
| +1 | Address of most recent RB |
| +9 | Address of most recent DEB |
| +13(D) | TIOT address |
| +16(1Ø) | Completion code |

2.5.6.28 Control Block Pointers. (Cont.)

| | |
|---|---|
| +25(19) | Address of most recent SPQE |
| +33(21) | Bit 7 Non-dispatchability bit |
| +37(25) | Address of most recent LLE |
| +113(71) | Address of first save area |
| +125(7D) | Address of TCB for job step task |
| +129(81) | Address of TCB for next subtask attached by same parent task |
| +133(85) | Address of TCB for parent task |
| +137(89) | Address of TCB for most recent subtask |
| +145(91) | Address of ECB to be posted at task completion |
| +153(99) | Address of dummy PQE minus 8 bytes |
| +161(A1) | Address of STAE control block |
| +181(B5) | Address of the job step control block |

UCB - Unit Control Block.

| | |
|---|---|
| -4 | CPU ID (used only with Model 65 Multiprocessing systems) |
| +2 | FF (UCB identification) |

2.5.6.28 Control Block Pointers. (Cont.)

| | |
|---|---|
| +4 | Device address |
| +13(D) | Unit name |
| +18(12) | Device class |
| +19(13) | Device type |
| +22(16) | Sense bytes (except devices with extended sense byte information) |
| +24(18) | Number of sense bytes (devices with extended sense byte information) |
| +25(19) | Address of sense bytes (devices with extended sense byte information) |
| +40(28) | Number of outstanding RESERVE requests (shared DASD only) |

2.5.6.29 OS/MVT Core Dump.

• This paragraph describes the format of an OS/MVT core dump. It is suggested that the dump be kept close at hand during the review of this analysis. See FIGURE 2-51 for explanation.

• The dump will be reviewed by going through certain portions of the dump and discussing the information provided.

• The first line of the dump (Circle A) displays the job name, step name, time of day (hhmmss where hh = hours, mm = minutes, ss = seconds), date (yyddd where yy = last two digits of year, ddd = day number of year), and page number. An ID field, not shown here, may be present if subtasks are used.

• The second line (Circle B) displays the completion code in hexadecimal notation, and indicates whether the SYSTEM (control program) or the USER (problem program) caused the termination.

• Line three (Circle C) displays the Program Status Word (PSW) AT ENTRY TO ABEND field, a field that is generally of little use in MVT dumps. The reason is that ABTERM enters an SVC 13 instruction in this PSW to branch to the ABEND routine (SVC 13) upon completion of its processing. For program checks, i.e.,

2.5.6.29 OS/MVT Core Dump. (Cont.)

errors caused under control of the Program Request Block (PRB), the right half of the PSW prior to entry to ABTERM is retained in the Active Program Status Word (APSW) field of the PRB. This area would, in that case, contain the address of the instruction to be executed following the program check.

- The Task Control Block (TCB) (Circle D) is shown for this active task; the register contents contained in the TCB are not shown for active tasks.

- The Request Block Pointer (RBP) field (Circle E) contains the address of the request block (RB) most recently added to the active RB queue.

- The DEB field (Circle F) points to the first data extent block for this task. This pointer is to the first byte of the DEB proper, following the prefix bytes. The first word of the DEB points back to the TCB address, the second word (Circle G) references the next DEB. Zeroes are shown in the address portion for the last DEB. The seventh word of the DEB (Circle H) proper points to the data control block (DCB) associated with this DEB.

- The TCB field points to the next lower priority task in the system. All TCB's are chained together on a main TCB queue which is used by the dispatcher to pass control to the highest dispatching priority TCB that can currently use the CPU resource.

- A field of zeros indicates that this task is currently the lowest priority TCB on this queue (MVT can dynamically create and delete TCB's, and the facility exists to alter dispatching priorities of TCB's, altering their effective positions on this main TCB queue).

- In the PRB, the first word is reserved (RESV). The second word, labeled APSW (Circle I), contains the right half of the user's PSW if the program interrupt occurred in the code represented by this RB (this did not occur in this example). This field would then contain the same information (i.e., the next instruction to be executed) that could be obtained from the PSW AT ENTRY TO ABEND field of other options of OS. Under MVT, this information is in the APSW field; the PSW AT ENTRY TO ABEND field is altered by ABTERM and is very rarely of help in debugging.

- The next field, FL-CDE (Circle J) contains a byte of flags pertaining to the way the module was obtained and the address of the contents directory element for the code controlled by this RB.

- The Wait-Link (WT-LNK) field (Circle K) indicates the number of requests waiting (the wait count) in the first byte; the LNK field points to the previous RB on the RB queue, or to the TCB if the RB containing the reference is the oldest RB.

- The SVRB format and some of its labeled fields differ from the fields of the PRB.

2.5.6.29 OS/MVT Core Dump. (Cont.)

• The APSW usually contains an identification of the SVC number in the high or low-order two bytes, depending on the SVC itself (this is actually the last first characters of the module name of the requested routine). For example, F5F5F1C4 (Circle L) represents SVC 55 or end-of-volume. F1FØF5C1 (Circle M) represents SVC 51 or ABDUMP. The PSW field of the preceding RB contains the hexadecimal representation of the SVC number in its fourth byte. Thus, in this dump, SVC 51 (Circle M) is shown as X'33' in the SVRB; SVC 13 is shown as X'OD'. Therefore, the PSW field provides a way to double check the SVC identification field shown in the APSW for SVRB's. The SVC 13 block will always contain the register contents, when available, for the user program.

• In the CDE section of the dump (Circle N), each contents directory element is listed separately.

• The EPA field (Circle O) defines the entry point address associated with the name in the NM field. For most ABEND conditions this is the address taken to compute the relative address, i.e., EPA address minus the APSW address will provide the relative address of Next Sequential Instruction (NSI).

• The XL/MJ field (Circle P), contains the starting address of the extent list (XL) for a major CDE, or the starting address of the major CDE if the module is a minor CDE (minor CDE's are created by alias entries or by the IDENTIFY macro which creates an additional entry point for a load module already in storage).

• Every major CDE contains an entry in the extent list which is identified by XL and begins immediately following the CDE's. (All CDE s were major CDE's on this dump.) The fields in the XL are:

SZ indicates the length of this entry in hexadecimal bytes.

NO shows the number of scattered control sections for the load module described by this entry (this is almost always 1, indicating block loading).

LN gives the length of the control section(s) of the load module defined by this entry. The high order bit is on if this is the last entry in the list.

ADR gives the starting address of the control section(s) defined by this XL entry.

Space for three LN-ADR entries exist on each line.

• Data extent blocks follow (Circle Q): A DEB exists for every data set open at the time of the dump. It generally contains information not available until the data set is opened. Each DEB is associated with a data control block (DCB) created as a part of the problem program. The DEB fields are not labeled

2.5.6.29 OS/MVT Core Dump. (Cont.)

as were the preceding control blocks of the dump; instead the entire DEB is presented unformatted. Specific fields useful in debugging will be identified below.

• The first word of the DEB points to the TCB address. The second word of the DEB addresses the next DEB on the chain for this task (the last DEB contains zeros). The ninth byte of the DEB contains a number of data set status flags of interest: Ø1 indicates that data set status (disposition) is OLD; 1Ø, status is MOD; 11, status is NEW. Three additional bytes follow which contain other flags.

• Offset 28 (1C hex), (Circle R) of the DEB contains a one byte field indicating the type of device dependent information beginning at offset 32 (2Ø hex). A Ø4 indicates direct access information, Ø2 is for non-direct access devices and communications devices. Unit record, magnetic tape, telecommunications and graphics devices are indicated with a Ø2. Indexed sequential and direct access sections are indicated with Ø4. The direct access section, applicable to this dump, defines the physical location of the data set on the direct access devices. Each extent entry contains 16 bytes and appears as shown below (Circle S). The first extent entry begins at offset 32 (hex 2Ø) of the DEB.

Direct access extent description segment:

| | File Mask | UCB Addr | BIN Num (2321) | Cyl Start Addr | Track Start Addr | Cyl End Addr | Track End Addr | No. of Tracks in Extent |
|---|---|---|---|---|---|---|---|---|
| Offset: | Ø | 1 | 4 | 6 | 8 | 1Ø | 12 | 14 |
| | 16 bytes | | | | | | | |

• The DCB for the data set is contained within the problem program and is referenced by the seventh word of the DEB. It is modified by OPEN and with the DEB and unit control block (UCB) serves as a repository of data management information pertinent to data sets used by the task. Although the DCB varies substantially based on the type of access method used, it generally consists of three segments: a device interface segment, a processing program interface segment (sometimes called the foundation segment) and an access method interface segment. The foundation segment is important in debugging -- it begins at offset 4Ø (28 hex), i.e., DCB address plus hex 28.

• Before OPEN, the foundation segment contains the DDNAME of the JCL statement defining the data set. This field is restored after the data set has been processed and CLOSE has been correctly completed.

2.5.6.29 OS/MVT Core Dump. (Cont.)

• While the data set is being processed, the first two bytes of the foundation segment field at offset 4Ø (28 hex) contains the Task Input/Output Table (TIOT) offset for the device entry of this data set (Circle T). The TIOT ties the job control DCB information to the unit control block (UCB) for this data set.

• If a complete (SYSABEND) dump is not available the DDNAME can still be found from the formatted TIOT in the dump. Find the TIOT offset from the DEB-DCB chain as described above. Subtract 4 from this figure in hex and then match this to the length identity of the entries in the formatted TIOT. The first is 14; others progress in increments of hex 14, i.e.,

| | Hex Identity |
|---|---|
| First entry | 14 |
| Second entry | 28 |
| Third entry | 3C |
| Fourth entry | 5Ø |
| Fifth entry | 64 |
| Sixth entry | 78 |
| Seventh entry | 8C |
| Eighth entry | A0 (etc.) |

The DDNAME is the second word of the entry found through this process.

2.5.6.29 OS/MVT Core Dump. (Cont.)

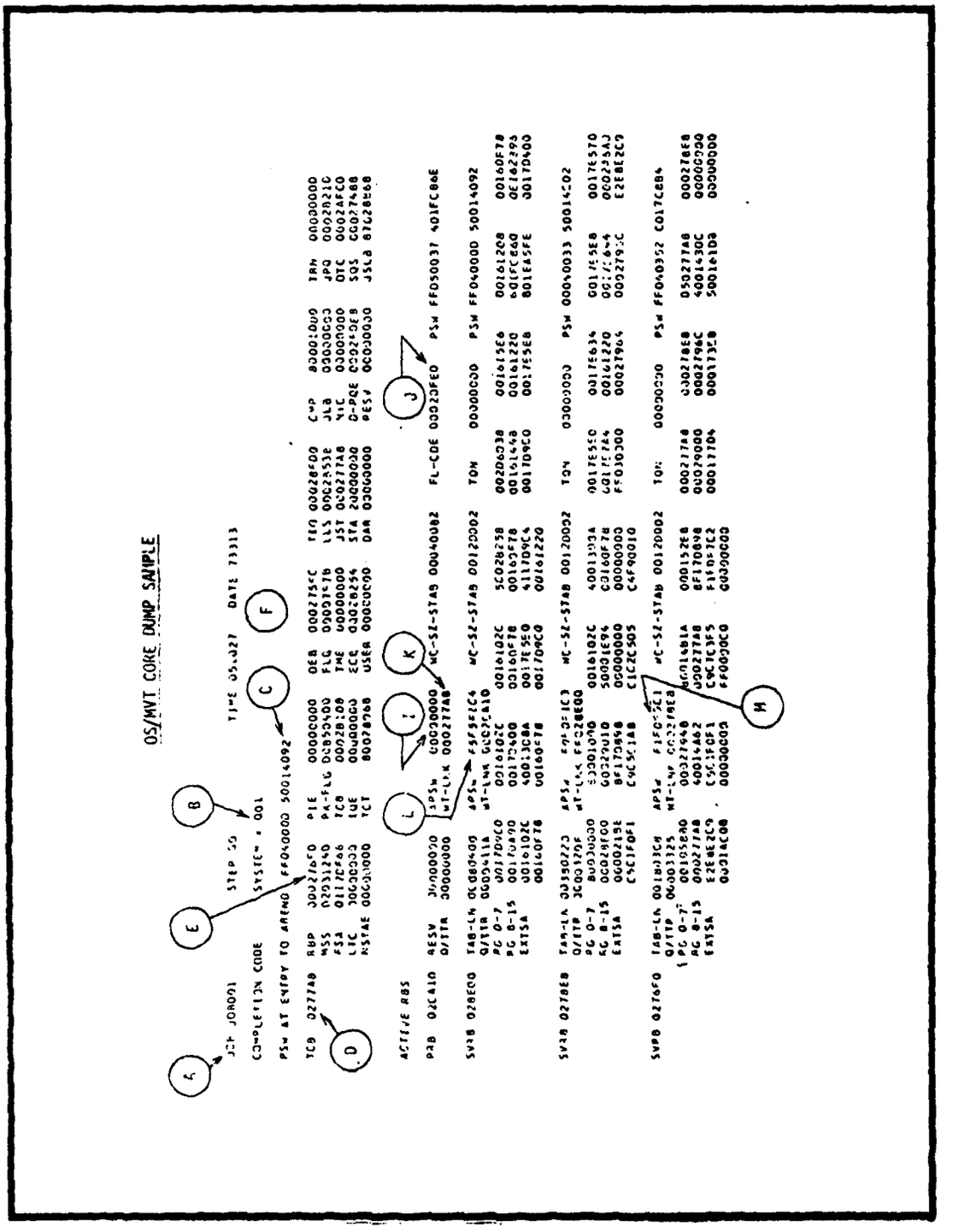



FIGURE 2-51

2.5.6.29 OS/MVT Core Dump. (Cont.)

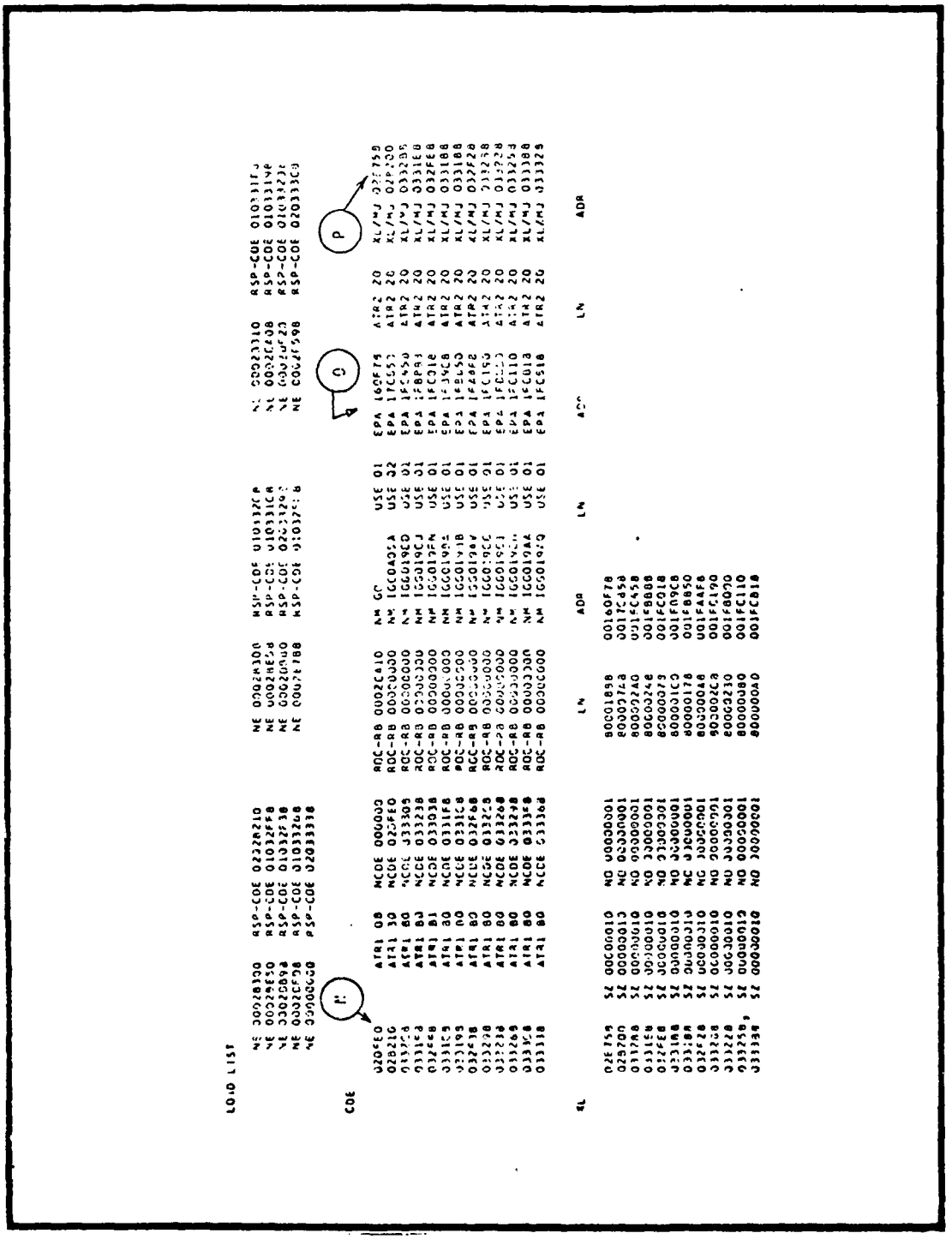


FIGURE 2-51 (Cont.)

2.5.6.29 OS/MVT Core Dump. (Cont.)

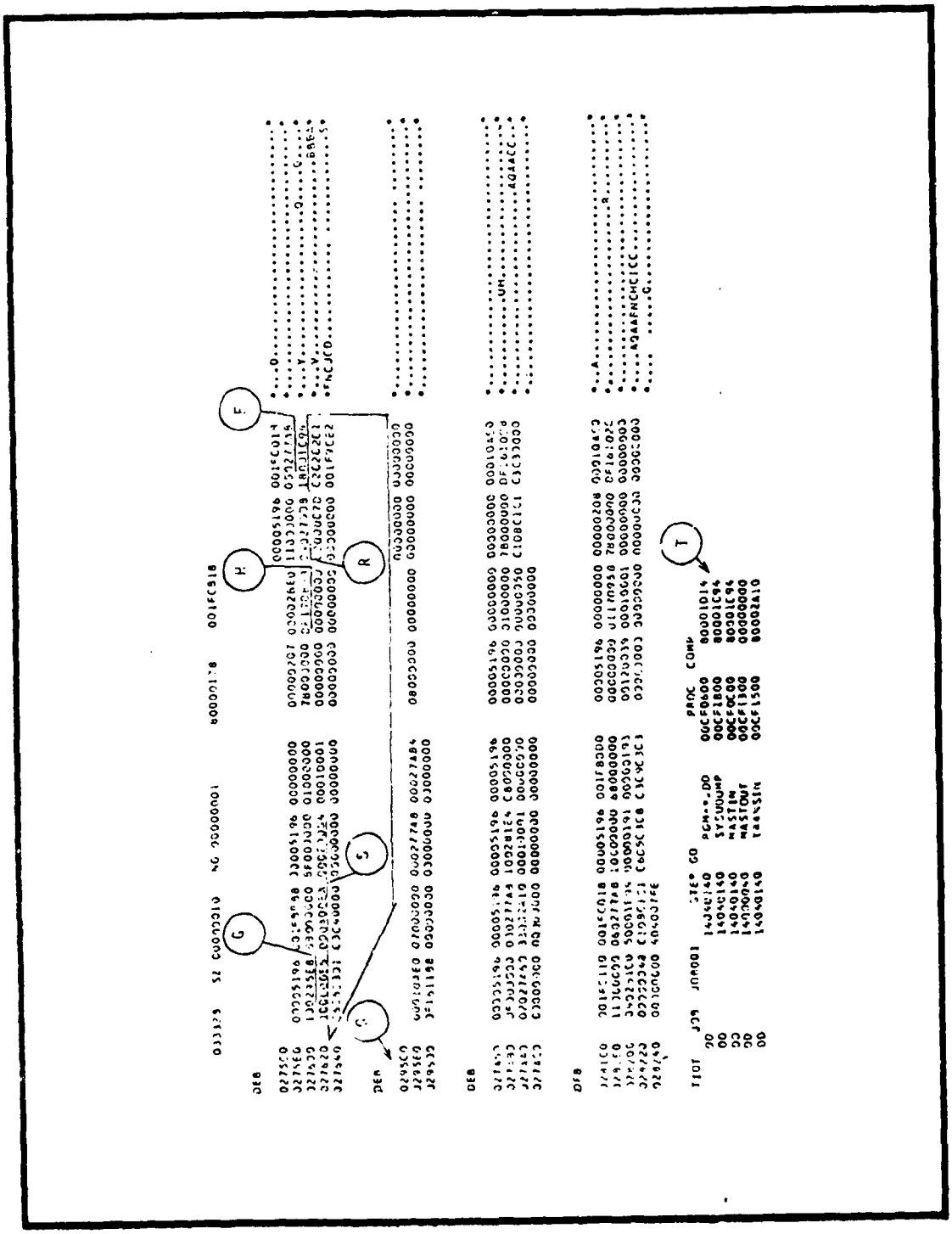
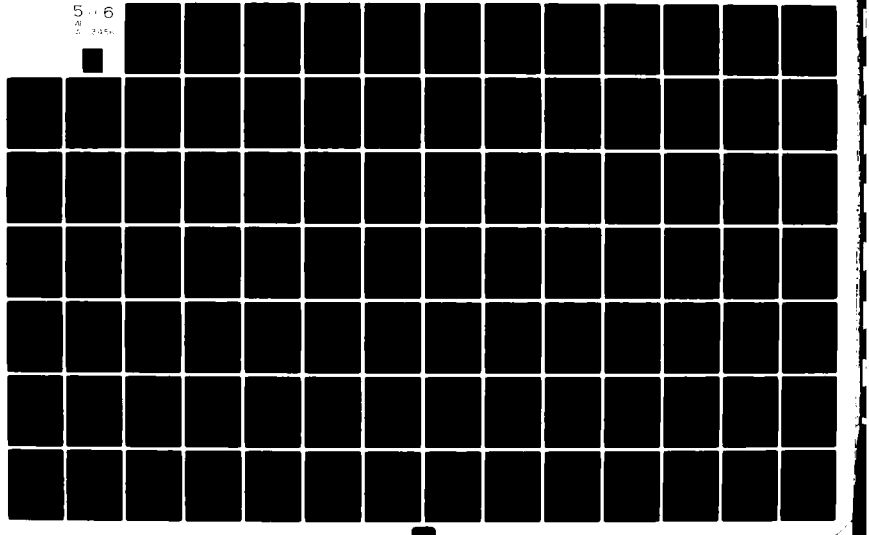
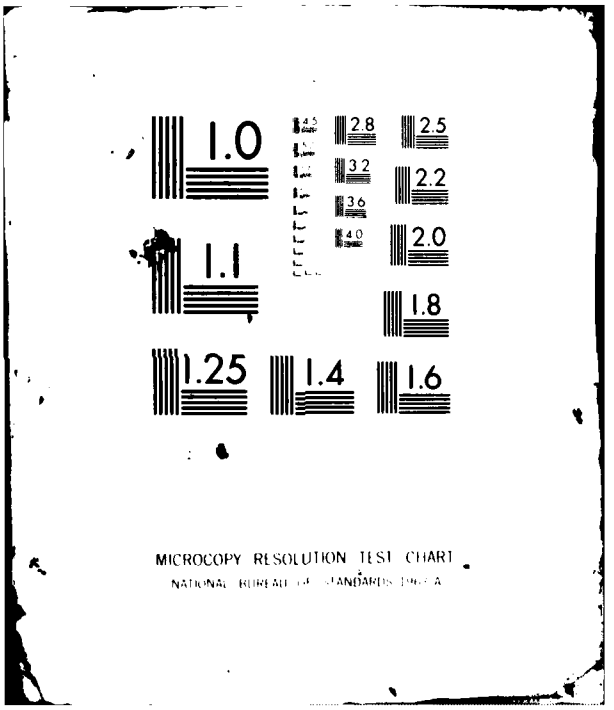



FIGURE 2-51 (Cont.)

AD-A113 456 ARMY COMPUTER SYSTEMS COMMAND FORT BELVOIR VA F/G 9/2
PROGRAMING PROCEDURES MANUAL (PPM).(U)
DEC 81

UNCLASSIFIED NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

2.5.6.30 OS Data Exceptions, Recognition and Error Recovery.

• This section is intended to assist the user in recognizing and correcting data exceptions which may occur in the program. It will show the user how to use an OS core dump to find the error in question and how to find the exact place in the program where the error occurred.

• The typical data exception results from the type of instruction that uses fields which are either defined or converted to the usage COMPUTATIONAL. The restrictions on such fields are as follows: The field must contain a numeric sign which is preceded by 1 to 31 decimal digits (Ø to 9 inclusive). This is true for both fields being used. The total length of a Storage to Storage (SS) instruction is six bytes and is broken down as follows: First byte operation code defining the instruction and one byte containing the hex value of the length of both fields minus one. The hex number on the left pertains to the length of the receiving field. The hex number on the right pertains to the length of the sending field. The next two bytes are the general register and displacement used to locate the receiving operand. The first hex digit indicates the base register. The three following hex digits represent the displacement. The last two bytes represent the sending field which is formatted the same as the receiving field, having a base register and a displacement.

• The fields are located in the core dump by taking the value that is found in the base register indicated and adding the displacement to that value in hex. The hex digit in the second byte as explained will give you the length of each field less 1 in bytes. Once the fields are found in the dump they should be examined for decimal digits and a sign of 'C', 'D', or 'F' in low order position.

• A data exception occurs when the sign is other than mentioned, or when a non-decimal digit appears in the field. As mentioned, by using the length you can determine where the digits and the sign should be. Both fields should be checked for errors. Once the field is found and determined to be either a sending or receiving field, the user should then locate the instruction that caused the exception in his compiled listing.

• To locate the offending instruction, subtract the program loading point (EPA), from the address of the error (APSW), which can be found on the first page of the dump. Once found, and reference is made to the PMAP, or CLIST, the line-position number will give you the exact COBOL verb used to generate the instructions. At this point, with this information, the Data Division must be checked for an invalid picture or value, if any. If there is a value, the data-name must be located everywhere in the program that it is used as a receiving field. Evaluate what is being put into the field. If the data-name is a group item, the sub-fields must be evaluated. In this case usually there are a number of signs in the core dump. Check all subroutines which are called using that data-name.

• Any field without a value clause contains the current value already in the machine. This data may be correct and it may not be correct. The use of the COBOL VERBS ON EXHIBIT and READY TRACE also are appropriate when trying to find program errors.

2.5.6.31 Register and Save Area.

• Review the register and save area conventions of the Operating System. Because of the large number of jobs run concurrently under OS, each program must take the precaution of saving the contents of the sixteen general registers for the program that called it. Then, by restoring those register contents before returning control, the program insures that the program that called it can continue to run successfully. Basic terminology for the dynamic linking of programs is:

The CALLING program is the program that calls a subprogram.

The CALLED program is the subprogram that is brought in by the calling program.

• In reality, the program is always a called program, since it is called in by the Operating System.

• If we then call in a subprogram, the main program is both a called and a calling program. It was called by the Operating System and it is calling by bringing in a subprogram.

• Therefore, each called program must, immediately upon gaining control, save the register contents of the calling program in the calling program's save area and restore the registers before turning control to the calling program.

• To do this, each program provides an eighteen word save area in which the registers will be saved by the called program. This is done automatically by the high level languages, but must be done by the programer in assembly language. Remember that the save area is provided by the calling program, but the registers are saved (stored in that save area) by the called program.

• The format of the save area is as shown in the following diagram:

| Word | |
|---|---|
| 1 | Used by PL/1 |
| 2 | Higher save area address (register 13) |
| 3 | Lower save area address |
| 4 | Register 14 (return address) |
| 5 | Register 15 (entry point address) |
| 6-18 | Registers Ø - 12 |

• Near the end of the formatted portion of an ABEND dump, is a section called the SAVE AREA TRACE. This portion of the dump traces the save areas used, first in forward sequence from the main program through the last subprogram called, then in backward sequence.

2.5.6.32 OC7 (Data Check) Debugging Exercise.

- The purpose of this section is to examine in detail an actual OC7 data check and analyze the conditions involved, both user created and system generated, which resulted with an ABEND and DUMP of a program written under OS.

- Before starting this debugging exercise, review FIGURE 2-52 through FIGURE 2-57 on the following pages which you will be requested to reference in detail.

- FIGURE 2-52 is a program listing of the Working-Storage Section and the Procedure Division of a COBOL program which, you as a programer, should be quite familiar with.

- FIGURE 2-53 is the Procedure Division Map (PMAP) which reflects:

The line number in the Procedure Division in which to reference an instruction.

The COBOL statement verb which begins the sentence being referenced.

The relative address within the user program where the object code is located.

The object code (machine language) representing the source statement as interpreted and coded by the COBOL compiler.

The assembler language coding representing the object code on the same line.

The identification of the FD statement within the Data Division, i.e., DCB-2 would reflect the FD as defined in the second SELECT statement in the ENVIRONMENT division. The BL notation (Base locator) will point to a register which contains the address of an instruction.

- FIGURE 2-54 and FIGURE 2-55 reflects the first and second pages of the core dump. Since the general format and description of a dump is explained in a previous section of this chapter, this section will address itself to only those areas that are necessary to successfully debug an OC7 (data check) ABEND.

- FIGURE 2-56 and FIGURE 2-57 formats the data and problem program areas of the core dump which we will reference in detail later on in this section.

- The first thing that should be done, upon receiving an ABEND, is to reference the appropriate completions code manual and make note of the probable causes that are applicable to the particular code that was given as a result of the ABEND.

2.5.6.32 OC7 (Data Check) Debugging Exercise. (Cont.)

- After considerable debugging experience, the meaning of the most common completion codes become second nature to the programer. At the beginning of this chapter, codes and their meanings are listed.

- The one ABEND condition that is singled out and addressed in this section is the OC7 (data check). An OC7 always indicates that the computer tried to perform an operation that turned out to contain invalid data or the data fields in question were not properly defined to accept the data given.

- At this point consider the OC7 ABEND that did take place (FIGURE 2-52 through FIGURE 2-57).

Reference FIGURE 2-54, block A, and note that the program terminated on the GO step, that the system completion code, block B, is an OC7.

FIGURE 2-54, block C, the Active Program Status Word (APSW) displays the address of the next sequential instruction (NSI) as located in the core dump. This is the NSI taken from the updated PSW after the interrupt occurred and a successful recovery not made by the interrupt handler. This is why the programer must remember to back up one instruction to reference the actual offending instruction. Make a pencil note of this APSW address (131874) because it is used later in determining the relative address of the NSI located in the PMAP.

FIGURE 2-55 block A, displaces the entry point address (EPA) which reflects the beginning address of the region where the problem program was loaded in main storage. Therefore, the EPA address is subtracted from the APSW address as follows:

```
  131874    APSW
- 1313CØ     EPA

    4B4     RELATIVE ADDRESS
```

The relative address, in this case is 4B4, is used to reference a point within the PMAP which provides assistance to the programer during most debugging exercises. (If the PMAP is not available, use the APSW address and go directly to core which will reference the Next Sequential Instruction.)

2.5.6.32 OC7 (Data Check) Debugging Exercise. (Cont.)

FIGURE 2-53, block A, reflects the NSI, but remember to back up one instruction to arrive at the actual instruction that ABENDED. Therefore, the relative address of the actual instruction is 4AE and the machine instruction on the same line is:

| Bits | 8 | 4 | 4 | 4 | 12 | 4 | 12 |
|---|---|---|---|---|---|---|---|
| | F9 | 1 | 1 | 6 | ØØØ | C | Ø38 |
| | OP Code | L1 | L2 | B1 | D1 | B2 | D2 |

Examination of the above instruction reveals that it is a storage to storage (SS) format and checking our data reference card (green card), we find that the F9 Operations Code (OP-Code) is a COMPARE DECIMAL operation. Further, we see that the length to each data field is two bytes long; the base register for the first operand (B1) reveals that general register 6 was used to store the base address for the first operand. Reference FIGURE 2-54, block D, which displays the contents of register 6 (1315AØ) plus the displacement for the first operand which, in this case is ØØØ, gives us the exact address in core of the data being operated on by the first operand. The same method should be used to find the data area for the second operand as follows:

FIGURE 2-53, block C.

Register C = 1317B8

plus displacement Ø38

1317FØ

The resulting value of 1317FØ give the exact location in main storage of data being operated on by the second operand.

Examination of core location 1315AØ, FIGURE 2-56, block A for two bytes, reveals the value of data as 483Ø (unsigned).

2.5.6.32 OC7 (Data Check) Debugging Exercise. (Cont.)

Examination of core location 1317FØ, FIGURE 2-57, block A, for two bytes, reveals the value of data as ØØ8ØC (signed).

Immediately we can see that we are attempting to add signed data to unsigned data. Further, we can determine that the field value at location 1315AØ is not properly formatted to contain valid numeric data and cannot be operated on under the current conditions.

Let's go to the Procedure Division and consider the logical aspects of the problem. FIGURE 2-53, block B, displays the system assigned line number in the Procedure Division where the source instruction may be found.

As indicated, FIGURE 2-52, block A, line 37 in the Procedure Division specifies that: IF SUB1 EQUAL ØØ8Ø GO TO IMAGE-END. In analyzing the statement we know that ØØ8Ø is a numeric literal which will be placed in packed, signed format by the compiler and treated so by a compare instruction. So, looking at the field as a separate entity, there is nothing wrong here.

The SUB1 data item that is being referenced would have to first be defined in the Data Division. FIGURE 2-52, block B, indicates that SUB1 was defined to accept three bytes of numeric data, packed and signed.

At first glance this may appear to be a valid way of defining data fields. But when you stop and realize that the S/36Ø does not clear storage unless told to do so and that it is the user's responsibility to initialize numeric data areas before they are needed for arithmetic operations, you will see that the data definition of SUB1 is not complete until the VALUE ZERO clause is added to the line. This is why the 483Ø value is still resident at location 1315AØ which is the data reference point in core for SUB1. Further, checking (data reference card) reveals that the value of 483Ø is actually part of a LOAD HALF (RX) machine language instruction and as far as the computer is concerned, the sign being absent, it was recognized as an alphabetic value. Thus, numeric data was to be compared to alphabetic data, which is illegal as far as the S/36Ø is concerned.

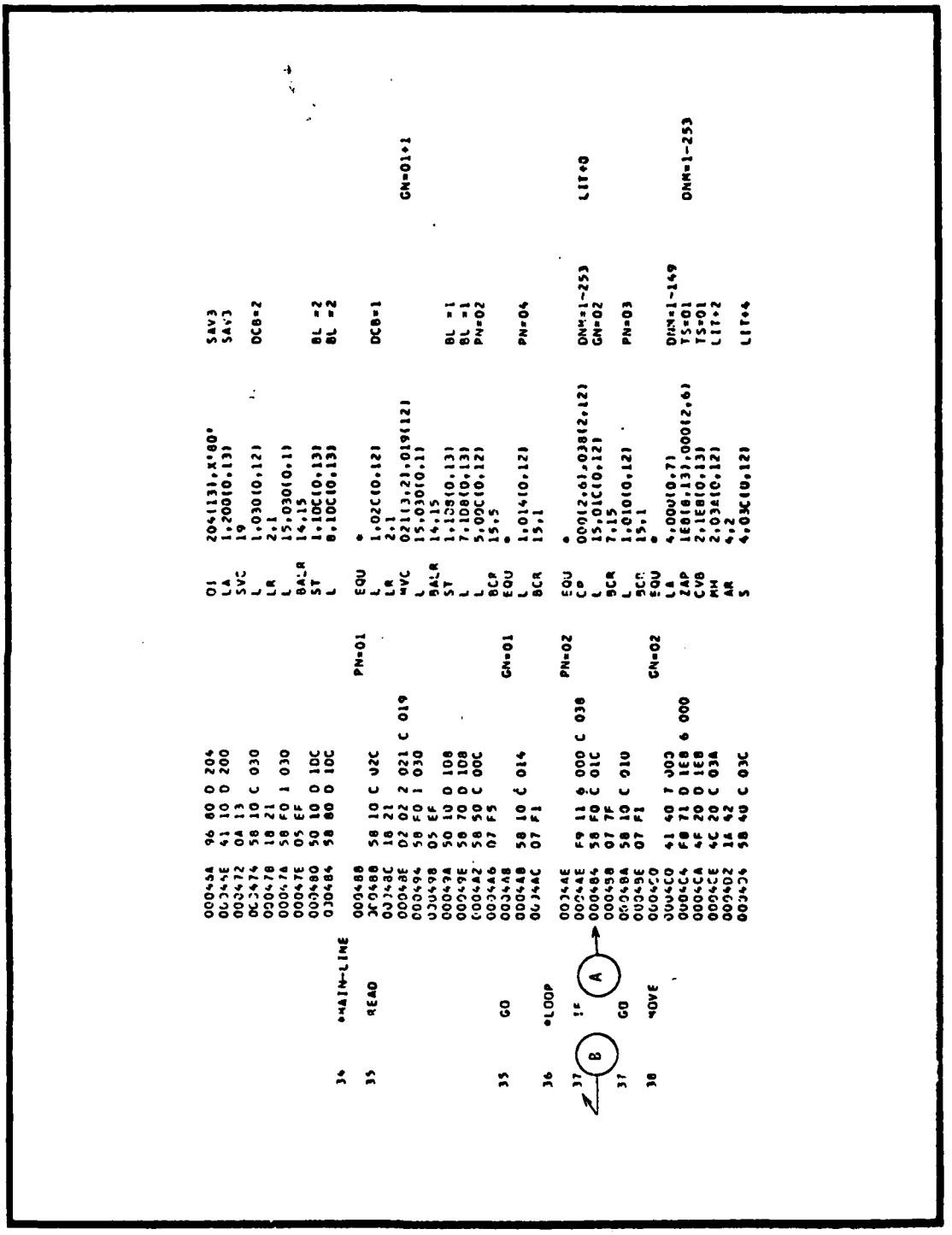
This is why the program ABENDED with a data check (OC7) and may be corrected by inserting the value clause on the SUB1 definition line.

2.5.6.32 OC7 (Data Check) Debugging Exercise. (Cont.)

```
00001          IDENTIFICATION DIVISION.
00002          PROGRAM-ID.   'REVERSE'
00003          ENVIRONMENT DIVISION.
00004          CONFIGURATION SECTION.
00005          SOURCE-COMPUTER.  IBM-370.
00006          OBJECT-COMPUTER.  IBM-370.
00007          INPUT-OUTPUT SECTION.
00008          FILE-CONTROL.
00009              SELECT IMAGE ASSIGN TO DA-3330-S-IMAGEIN.
00010              SELECT IMAGET ASSIGN TO DA-3330-S-IMAGEOUT.
00011          DATA DIVISION.
00012          FILE SECTION.
00013          FD  IMAGE,
00014              LABEL RECORDS ARE OMITTED,
00015              RECORDING MODE IS F,
00016              DATA RECORD IS INPUT-IN,
00017              BLOCK CONTAINS 0 RECORDS.
00018          01  INPUT-IN.
00019              03 DATA-IN OCCURS 80 TIMES PICTURE X.
00020          FD  IMAGET,
00021              LABEL RECORDS ARE STANDARD,
00022              RECORDING MODE IS F,
00023              BLOCK CONTAINS 5 RECORDS,
00024              DATA RECORD IS OUTPUT-OUT.
00025          01  OUTPUT-OUT.
00026              03 OUTPUT-OT OCCURS 80 TIMES PICTURE X.
00027          WORKING-STORAGE SECTION.
00028          01  WS-SUBSCRIPTS                    COMP-3.
00029      (B)-->  03  SUB1                         PICTURE S999.
00030              03  SUB2                         PICTURE S999.
00031          PROCEDURE DIVISION.
00032          TOPPER.
00033              OPEN INPUT IMAGE, OUTPUT IMAGET.
00034          MAIN-LINE.
00035              READ IMAGE, AT END GO TO JOB-END-ROUTINE.
00036          LOOP.
00037              IF SUB1 = 80 GO TO IMAGE-END.
                      (A)
```

FIGURE 2-52

2.5.6.32 OC7 (Data Check) Debugging Exercise. (Cont.)



```
                                                
```

2.5.6.32 OC7 (Data Check) Debugging Exercise. (Cont.)

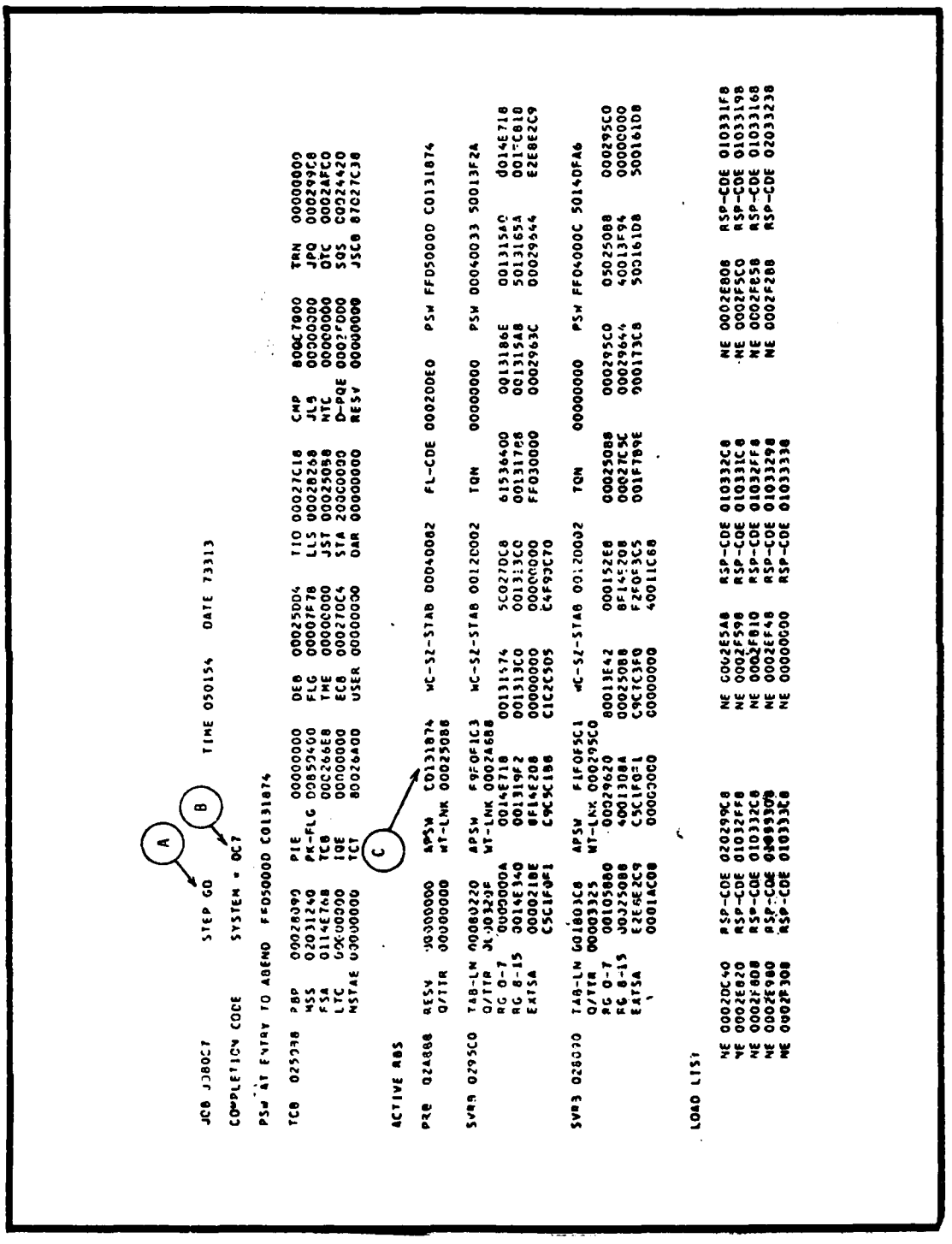



FIGURE 2-54

2.5.6.32 OC7 (Data Check) Debugging Exercise. (Cont.)

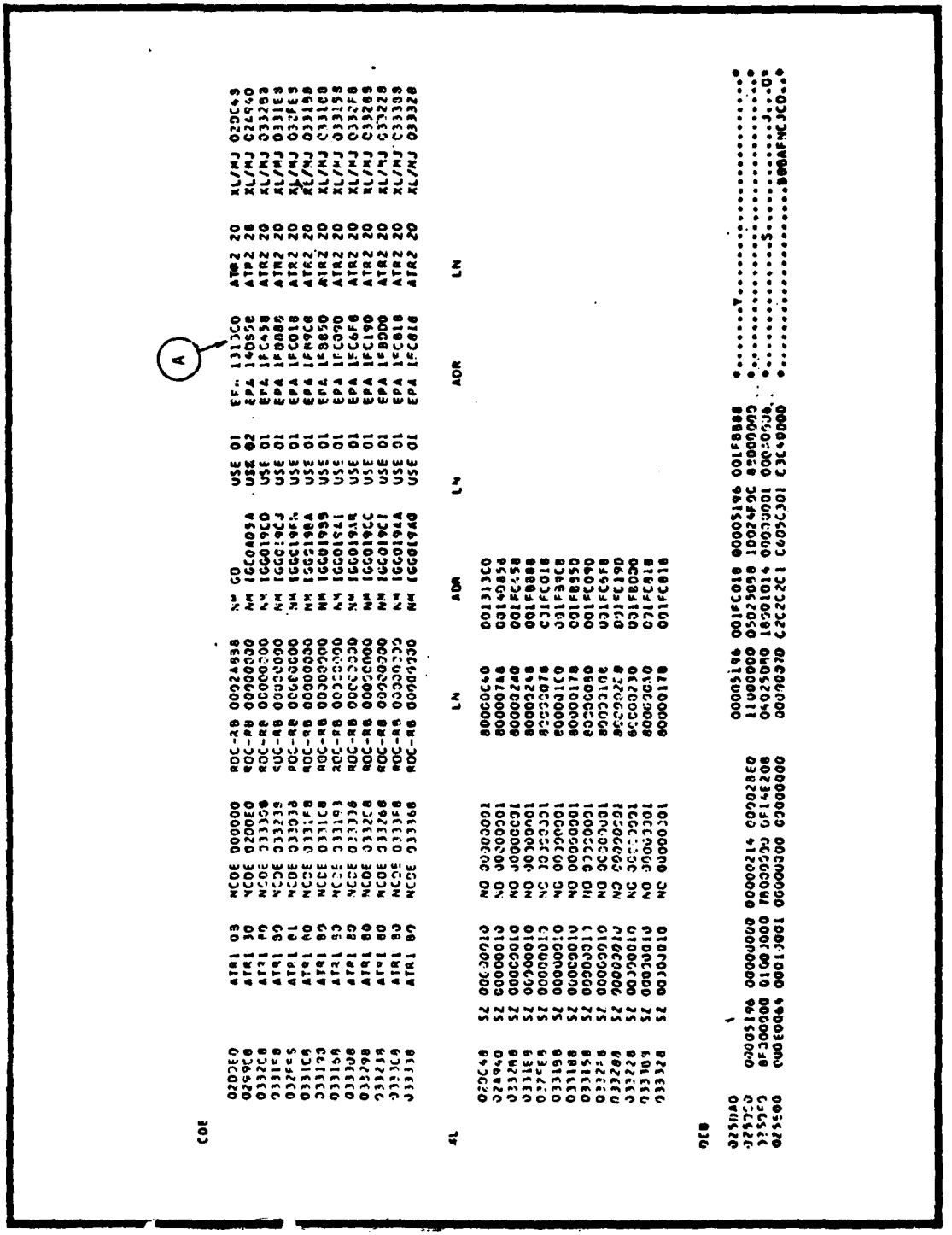


FIGURE 2-55

2.5.6.32 OC7 (Data Check) Debugging Exercise. (Cont.)

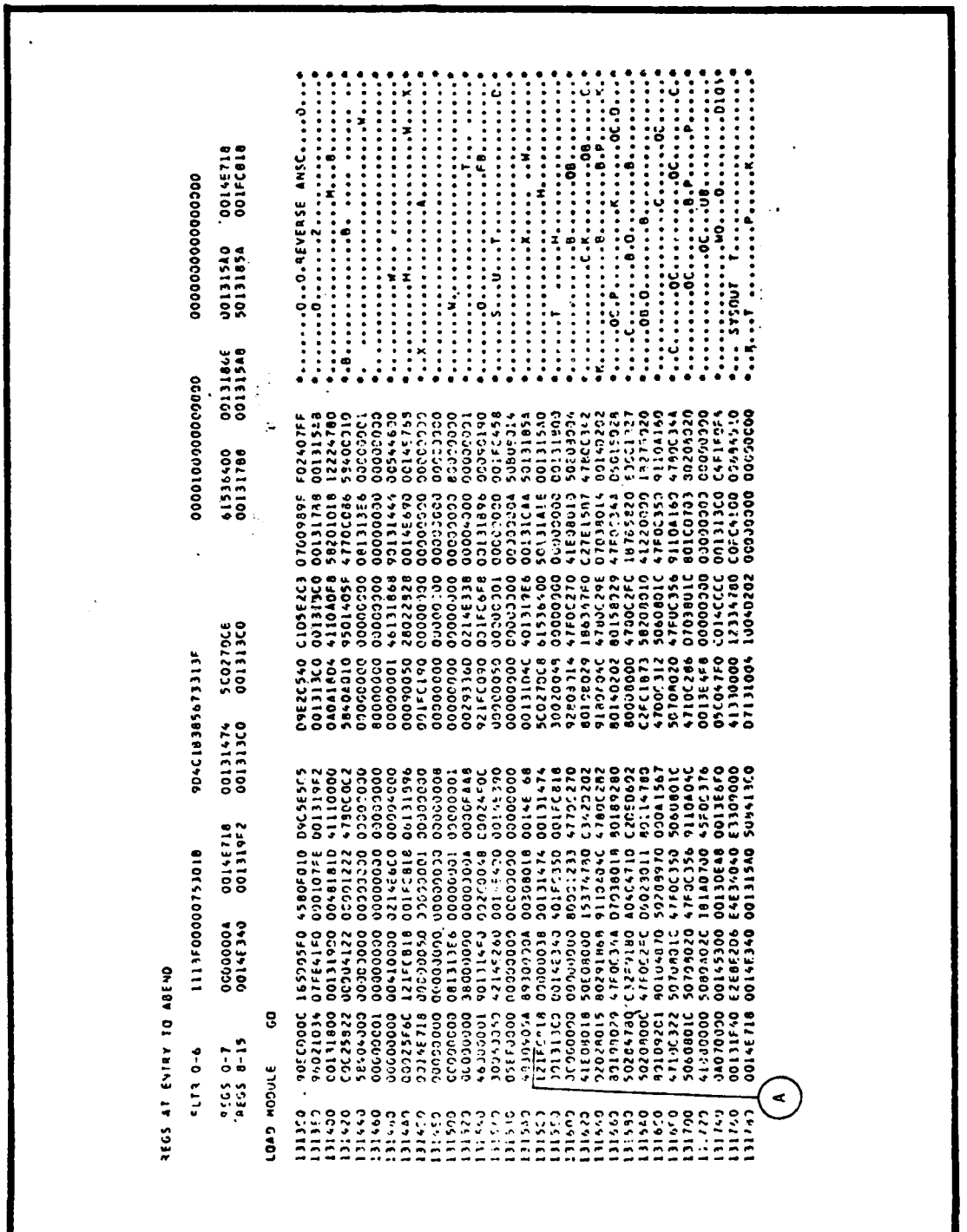


FIGURE 2-56

2.5.6.32 OC7 (Data Check) Debugging Exercise. (Cont.)

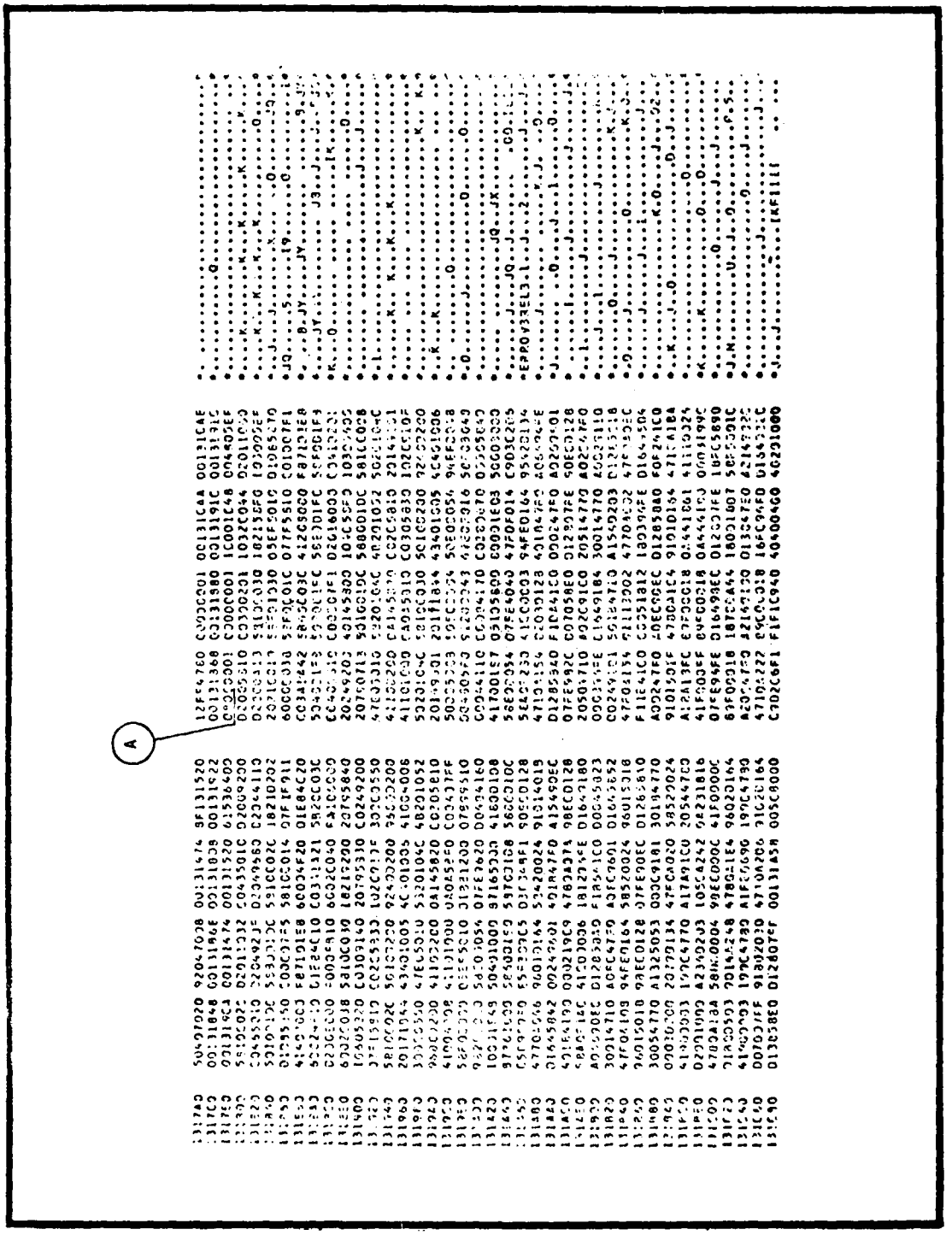



FIGURE 2-57

2.5.6.33 Debugging of COBOL Segmented Programs Under OS/MFT.

- This paragraph is designed to assist programers in debugging segmented programs. Special techniques are necessary to determine which segment is in core at the time a segmented program abends, and to find the program location of the last instruction executed.

- The link edit map and core dump must be available.

- Steps to be followed for debugging a segmented program.

Determine the origin of the abending program by subtracting the segment table length (FIGURE 2-58) from the user entry point given in the first active RB in the core dump (FIGURE 2-59).

```
1D88Ø     User Entry Point
- 38      Segment Table Length

1D848     Origin of Load Module
```

Subtract the origin of the abending program from the current PSW address (FIGURE 2-59). This gives the address of the abend relative to zero:

```
 22786    Current PSW Address
-1D848    Origin of Load Module

 4F3E     Address of Abend Relative to Zero
```

If the address relative to zero is within the root segment (less than the origin of the overlay area), continue with normal debugging procedures. Otherwise, the address relative to zero should point to a location within the overlay area. The current segment in core must be determined by adding the CURSEGM location (FIGURE 2-58) to the load module origin. This gives the location of the segment priority number.

2.5.6.33 Debugging of COBOL Segmented Programs Under OS/MFT. (Cont.)

| | |
|---|---|
| 1D848 | Origin of Load Module |
| +3611 | CURSEGM Location |
| 2ØE59 | Location of Segment Priority Number |

The segment priority number is a binary half word. In the example, the priority number is X'37' (FIGURE 2-59) or decimal 55. The priority number in decimal will identify the segment in core. This priority number corresponds to the priority assigned to the sections of the COBOL source program.

The address of the failing instruction, relative to the beginning of the overlay segment, is calculated by subtracting the origin of the overlay area (FIGURE 2-58) from the address of the abending instruction relative to zero:

| | |
|---|---|
| 4F3E | Address of abend relative to zero |
| -4298 | Origin of Overlay Area |
| CA6 | Address of Abend Relative to beginning of overlay segment |

The programer can now refer to the compile source listing for the overlay segment in error and determine the failing instruction as in normal debugging procedures.

2.5.6.33 Debugging of COBOL Segmented Programs Under OS/MFT. (Cont.)

SAMPLE LINK EDIT MAP

```
F128-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED XREF, LIST, LET,
   DEFAULT OPTION(S) USED - SIZE = (151552,4Ø96Ø)

IEWOOO                INSERT P3MALL
IEWOOO                OVERLAY A
IEWOOO                INSERT P3MALL5Ø
IEWOOO                OVERLAY A
IEWOOO                INSERT P3MALL55
IEWOOO                OVERLAY A
IEWOOO                INSERT P3MALL6Ø
IEWOOO                ENTRY P3MALL
```

CROSS REFERENCE TABLE

| CONTROL SECTION | | | | ENTRY | |
|---|---|---|---|---|---|
| NAME | ORIGIN | LENGTH | SEG.NO. | NAME | LOCATION |
| $SEGTAB | ØØ | 34 | 1 | | |
| P3MALL | 38 | 2Ø7A | 1 | | |
| ILBODTEO | 2CB8 | 7ØØ | 1 | | |
| | | | | ILBODTE1 | 33C2 |
| ILBOSGMO | 34E8 | 12F | 1 | | |
| | | | | CURSEGM | 3611 |
| P3MALL5Ø | 4298 | 4BE | 2 | | |

LOCATION REFERS TO SYMBOL IN CONTROL SECTION SEG.NO.

SAMPLE LINKEDIT MAP

| CONTROL SECTION | | | | ENTRY | |
|---|---|---|---|---|---|
| NAME | ORIGIN | LENGTH | SEG.NO. | NAME | LOCATION |
| P3MALL55 | 4298 | 3994 | 3 | | |

LOCATION REFERS TO SYMBOL IN CONTROL SECTION SEG.NO.

| CONTROL SECTION | | | | ENTRY | |
|---|---|---|---|---|---|
| NAME | ORIGIN | LENGTH | SEG.NO. | NAME | LOCATION |
| P3MALL6Ø | 4298 | F4 | 4 | | |

LOCATION REFERS TO SYMBOL IN CONTROL SECTION SEG.NO.

FIGURE 2-58

2.5.6.33 Debugging of COBOL Segmented Programs Under OS/MFT. (Cont.)

SAMPLE CORE DUMP

```
JOB P3MALL3T STEP STEP2 TIME 1Ø54Ø4 DATE 74169

COMPLETION CODE SYSTEM = OC7

PROGRAM INTERRUPTION (DATA) AT LOCATION Ø2278Ø

INTERRUPT AT Ø22786

PSW AT ENTRY TO ABEND FF75ØØØD CØØ22786

TCB  OOBDCO  RB  OOO35458  PIE  ØØØØØØØØ  DEB  ØØØ355CC  TIOT  ØØØ3667Ø
             MSS ØØØØBE9Ø  PK  7ØA1ØØØ8  FLG  ØØØØØ4AD  LLS   ØØØ35Ø58

ACTIVE RBS

PRB  Ø1D8ØØ  NM P3MALL SZ/STAB ØF8FØØCØ USE/EP ØØØ1D88Ø   PSW
             FF75ØØØD CØØ22786

LOAD LIST

LPRB Ø3628Ø nM IEWSZOVR SZ/STAB ØØ382Ø1Ø USE/EP Ø2Ø3627AØ

GA Ø-7 ØØØØØØFF ØØØØ52BØ ØØØØ52Ø8 ØØØØ56A9 ØØØØØØØØ ØØØØØØØØ
       ØØØØØØØØ ØØØØØØØØ

GR 8-F ØØØØ581A 8AØØ6632 ØØØØ6632 ØØØØ5Ø2A ØØØØ5FAØ ØØØØØØØØ
       ØØØØØ35Ø ØØØØØØØØ

ØØØØØØ ØØØØØØØØ ØØØØØØØØ ØØØØØØØØ ØØØØØØØØ ØØØØØØØØ ØØØØØ35Ø
       FFØ6ØØ4Ø ØØØØØØØØ

ØØØØ1Ø FFØ5ØØØ7 4ØØØ483E ØØØØØØØØ ØØØØØØØØ 5B58C2C5 ØØØØØØØØ
       FFØ5ØØEØ 7ØØØ474A

Ø2ØE4Ø 9640F126 132247FØ FØØ4ØØ2B Ø8ØØ5169 ØØØØØØ7F 7FFFFFFF
       ØØ37ØØ37 ØØ37ØØ3Ø
```

FIGURE 2-59

## 2.6 USACSC COBOL PROGRAM DESIGN TECHNIQUES.

• While most programing problems defy a common approach, a number of general techniques can aid in writing, debugging and maintaining computer programs. This section recommends approaches to programing which, in the long run, will free the programer to put more time and energy into problem-solving and responding to users' needs.

### 2.6.1 PROCEDURE DIVISION DESIGN.

THE MAINLINE.

• The mainline should be devoted to the chief processing activity. Implied in this is the logical interface function. Choosing what problem-solving code is to be executed is its main function. This control is executed through testing then performing subroutines. If abnormal situations can arise, a cursory test can be made to identify the cases and then be dealt with separately in a subroutine.

• The mainline should describe the program in its most abstract form. A programer should be able to pick up a program listing and, by reading through the mainline, gain a general knowledge of the program's major functions.

Initial housekeeping.

Subroutines.

End-of-job processing.

HOUSEKEEPING.

• Housekeeping is a familiar function to everyone. The obvious tasks included in housekeeping are opening of files, data field initialization, obtaining the date and/or time, table loading, etc.; in short, anything that deals more with preparing to run a program than accomplishing the program's functions.

• Most housekeeping is performed once, such as file opening; some is performed cyclically such as blanking fields after each transaction. These one-time functions can be kept separate so that the cyclical occurring actions can be performed when required.

• All housekeeping routines should be PERFORMed by the mainline of the program.

SUBROUTINES.

• A subroutine consists of one, clear and well-defined function.

• The PERFORM verb and paragraph structure of COBOL provide the natural environment for closed subroutine structures.

2.6.1 PROCEDURE DIVISION DESIGN. (Cont.)

• Since COBOL is not strict in its structure of the PERFORM statement, the user can easily violate its usefulness in implementing closed subroutines. Therefore, as mentioned under the PERFORM verb in the PROCEDURE DIVISION Techniques, the PERFORM verb should always be used with the following format:

```
PERFORM paragraph-name THRU paragraph-exit-name.
```

Paragraph-exit-name should be an EXIT paragraph. By using the THRU option, the scope of the code referenced by the PERFORM is clearly indicated.

• PERFORMing a SECTION is not a good technique because the tendency is to forget to define the scope of a SECTION by another SECTION statement.

• A subroutine should always have a single entry point and a single exit point. All conditional and unconditional branches within the subroutine should always pass control to the EXIT paragraph. The point of entry of a subroutine should always be the same. The user can probably see little harm in PERFORMing an inner paragraph. However if this rule is violated it can at some time cause program difficulties.

• Subroutines should be grouped together in the program apart from the main line or process-directing paragraphs. These process-directing paragraphs should never be branched to (GO TO ...) by a subroutine. A GO TO, AT END path, etc., should never cause control to pass from the range of a PERFORM.

• Entire paragraph-groups can be frequently reused in more than one program. If a given paragraph-group is often executed it may be more desirable to include it in-line rather than using it in a called subprogram.

• The COPY facility of COBOL provides the programer with the capability of reusing code.

END-OF-JOB ROUTINES. End-of-job routines normally include closing of files and either return to the mainline routine or STOP RUN.

2.6.2 STANDARD LOGIC CONSTRUCTS.

CONDITIONAL STATEMENT ORDERING.

• The relative frequency of data item values should be considered in ordering conditional statements within the source program.

2.6.2 STANDARD LOGIC CONSTRUCTS. (Cont.)

• EXAMPLE. Program A (FIGURE 2-60) is a master file update accepting five basic transactions, each with the following occurrence ratio:

$$\frac{\text{number of one transaction type}}{\text{TOTAL NUMBER TRANSACTIONS type}} :$$

| TRANSACTION CODE | MEANING | FREQUENCY |
|---|---|---|
| A | Add record to file | 40% |
| C | data field change | 15% |
| D | delete record from file | 30% |
| K | control field change | 2% |
| I | inquiry | 13% |

FIGURE 2-60

Therefore within the source program the tests of transaction code should occur as follows: (Refer to FIGURE 2-61.)

2.6.2 STANDARD LOGIC CONSTRUCTS. (Cont.)

```
ØØ1Ø-DIRECT-TRANS.
     IF T-TRANS-CODE IS EQUAL TO 'A'
          PERFORM ØØ6Ø-ADD-MASTER THRU ØØ7Ø-ADD-MASTER-X
          GO TO ØØ5Ø-READ-TRANS.
     IF T-TRANS-CODE IS EQUAL TO 'D'
          PERFORM ØØ8Ø-DELETE-MASTER THRU ØØ9Ø-DELETE-MASTER-X
          GO TO ØØ5Ø-READ-TRANS.
     IF T-TRANS-CODE IS EQUAL TO 'C'
          PERFORM Ø1ØØ-CHANGE-MASTER THRU Ø11Ø-CHANGE-MASTER-X
          GO TO ØØ5Ø-READ-TRANS.
     IF T-TRANS-CODE IS EQUAL TO 'I'
          PERFORM Ø12Ø-INQUIRY-EXTRACT THRU Ø13Ø-INQUIRY-EXTRACT-X
          GO TO ØØ5Ø-READ-TRANS.
     IF T-TRANS-CODE IS EQUAL TO 'K'
          PERFORM Ø14Ø-CONTROL-CHANGE THRU Ø15Ø-CONTROL-CHANGE-X
          GO TO ØØ5Ø TO ØØ5Ø-READ-TRANS.
```

FIGURE 2-61

LATEST DECISION PRINCIPLE

- This principle asserts that decisions which control program flow should not be made now if they can be made at a later time. This action usually results in a program which is easy to follow, and it generally eliminates many redundant statements.

2.6.2 STANDARD LOGIC CONSTRUCTS. (Cont.)

- Example:

INCORRECT APPROACH

```
        IF A IS EQUAL TO B
             GO TO ØØ4Ø-XY.
        MOVE C TO X.
        MOVE D TO Y.
        GO TO ØØ6Ø-ZZ.
    ØØ4Ø-XY.
        MOVE C TO X.
        MOVE E TO Y.
    ØØ6Ø-ZZ. MOVE F TO Z.
```

LATEST DECISION APPROACH

```
        MOVE C TO X.
        MOVE F TO Z.
        IF A IS EQUAL TO B
             MOVE E TO Y
          ELSE
             MOVE D TO Y.
```

2.6.2 STANDARD LOGIC CONSTRUCTS. (Cont.)

LOOP CONTROLLING.

• The most natural and apparent loop in COBOL is the PERFORM with the VARYING option. This form of looping, combined with indexing, can eliminate most of the typical problems such as underflow and overflow of the index.

• The general ANSI construct of this standard form is:

```
PERFORM paragraph-name THRU paragraph-exit-name
VARYING index-name FROM identifier-1 BY
identifier-2 UNTIL condition-statement
```

The index-name should be unique for each PERFORM ... VARYING.

If care has been taken to define a simple, one-dimensional table, the initial value (identifier-1) and the stepping value (identifier-2) can usually both have a value of 1.

The condition statement tests the index-name for its having equaled a predetermined limit.

Example:

```
PERFORM IA-MARCH-LOOP THRU LAX-MARCH-LOOP-X
     VARYING MAR-INDEX FROM 1 BY 1
     UNTIL MAR-INDEX IS GREATER THAN 31.
```

• It cannot be overstressed that much difficulty may be avoided by using the PERFORM ... VARYING for looping. There may be a few instances, however, in which it is not feasible to use this construct. In these cases, every step of a loop should be clearly separated and defined.

```
INITIALIZE.

INCREMENT.

TEST INDEX-VALUE.

TEST CONDITION.

EXIT.
```

A practical method of programing this type of situation is to first program the body of the loop for the general case and next program it for the last iteration. Then backup and program the initialization and incrementation step. Finally, determine if the body will operate correctly for all values of the parameter.

2.6.2 STANDARD LOGIC CONSTRUCTS. (Cont.)

Making it simple to find the whereabouts of looping tasks (initializing, incrementing, testing, etc.) can bring increased success. If there is good reason why these tasks must be separated, then careful structural arrangement of the program can help increase the readability. For example, if involved computation is required to obtain values assigned to an index, limit and increment value, PERFORM the module(s) to determine these parameters just prior to PERFORMing the process module.

Example: programer control of looping with indexing.
(Refer to FIGURE 2-62.)

This example shows a practical method of loop controlling. The programer should note, however, that the PERFORM ... VARYING could accomplish the function as well.

```
.
.
.
WORKING-STORAGE SECTION.

77      WS-INDEX-LIMIT          USAGE IS INDEX.
.
.
.
PROCEDURE DIVISION.

ØØ1Ø-HOUSEKEEP.

          SET WS-INDEX-LIMIT TO 32.
.
.
.
ØØØ2Ø-AA-ENTER.

          SET WS-A-INDEX TO ZERO.

ØØ3Ø-AB-LOOP.

          SET WS-A-INDEX UP BY 1.

          IF WS-A-INDEX-LIMIT IS EQUAL TO WS-INDEX-LIMIT

               GO TO ØØ4Ø-ABX-LOOP-EXIT.

          GO TO ØØ3Ø-AB-LOOP.

ØØ4Ø-ABX-LOOP-EXIT.

          EXIT.
```

FIGURE 2-62

2.6.2 STANDARD LOGIC CONSTRUCTS. (Cont.)

RELATION CONDITIONS--TESTING THE INDEX.

• When testing an index-name used in a loop for its limit, a data conversion may be necessary for certain comparisons. In addition, some comparisons are illegal. The following rules apply:

1. The comparison of two index-names is actually the comparison of the corresponding occurrence numbers.
2. In the comparison of an index-name with an ordinary data item or with a literal, the occurrence number that corresponds to the value of the index-name is compared with the data item or literal.
3. In the comparison of an index data item with an index-name or another index data item, the actual values are compared without conversion.
4. Any other comparison involving an index data item is illegal.

INDEX-NAMES AND INDEX DATA ITEMS--COMPARISONS

| SECOND OPERAND / FIRST OPERAND | INDEX-NAME | INDEX DATE ITEM | DATE-NAME (numeric integer) | LITERAL (numeric integer) |
|---|---|---|---|---|
| INDEX-NAME | Compare occurrence number | Compare without conversion | Compare occurrence number with data-name (conversion implied) | Compare occurrence number with literal (conversion implied) |
| INDEX DATA ITEM | Compare without conversion | Compare without conversion | Illegal | Illegal |
| DATE-NAME (numeric integer) | Compare occurrence number with data-name (conversion implied) | Illegal | | |
| LITERAL (numeric integer) | Compare occurrence number with literal (conversion implied) | Illegal | | |

FIGURE 2-63

2.6.2 STANDARD LOGIC CONSTRUCTS. (Cont.)

USING FLAGS.

• The term "flag" refers to the use of data items to control processing. "Switch" can be a synonym, however, it has a second generation-computer connotation of toggle switches on the computer's console. The use of a flag is simple. Somewhere in the program a MOVE statement sets the value of the flag. Elsewhere in the program the value of the flag is tested and action is taken dependent on the value. Finally the flag may be reset.

• The use of flags can be a problem area.

The programer can lose track of where in the logic the flag was last set or reset.

The same flag is used for several purposes, and again the programer loses track of which use last set the flag's value.

• The use of flags/switches is not recommended and programers are advised to develop a habit of using data to determine proper logic flow.

The technique of moving (HIGH-VALUES) to an input (WORKING-STORAGE) area when encountering an end-of-file condition provides a logical means of determining which file should be read.

In the event flags/switches are required the important point is to develop consistency. If a programer treats flags/switches in the same manner each time a high level of dependability results. Keep flag/switch constructs simple and visible.

Comment lines both within the logical flow and the data description of flags add further clarity.

Alphanumeric program flags result in more efficient object code when condition testing is done. For simple program flags, there are certain techniques applicable across vendor lines.

1. For improved readability and maintainability of program flags, it is best to use alphanumeric flags with values of "Y" or "N" to indicate the field status. This can be done in two ways. A data item can be interrogated directly by data-name or condition-names (88-levels) may be assigned and interrogated. (Refer to FIGURE 2-64.)

2.6.2 STANDARD LOGIC CONSTRUCTS. (Cont.)

```
WORKING-STORAGE SECTION.

77  WS-EOJ-FLAG                PIC X VALUE "N".
    .
    .
    .
    Ø5  WS-FILE-STATUS          PIC X VALUE "Y".
88  FILE-OPEN VALUE IS "Y".
88  FILE-CLOSE VALUE IS "N".
    .
    .
    .
```

FIGURE 2-64

2. The use of an alphanumeric PICTURE X rather than numeric PICTURE 9 for flags has definite efficiency considerations for some vendors. For all vendors, the alphanumeric flag is more desirable because of readability.

2.7 COBOL PROGRAM STRUCTURE TECHNIQUES.

WORKING STORAGE ORGANIZATION.

- To help improve source listing readability and debugging capability, the following formatting techniques are recommended for the WORKING-STORAGE SECTION. This organization will result in execution-time improvement and main-storage savings for some vendors.

- The placement of the following non-numeric literals as the first and last WORKING-STORAGE statement is an aid in locating this section of the program in object-time dumps.

```
77  FILLER                     PICTURE X(44) VALUE
        "PROGRAM XXXXXXXX WORKING-STORAGE BEGINS HERE".

Ø1  FILLER                     PICTURE X(42) VALUE
        "PROGRAM XXXXXXXX WORKING-STORAGE ENDS HERE".
```

2.7 COBOL PROGRAM STRUCTURE TECHNIQUES. (Cont.)

• These two literals will appear in all dumps of the program delineating the WORKING-STORAGE SECTION. The program-name replaces the XXXXXXXX in the literal.

• The actual WORKING-STORAGE data should be divided into three groups: COMPUTATIONAL 77-level items, all other 77-level items and Ø1-level items. The structure within groups is listed below.

COMPUTATIONAL 77-level items should have the following structure:

Items with PICTUREs from S9(1Ø) through S9(18), in alphabetical order.

Items with PICTUREs from S9(5) through S9(9), in alphabetical order.

Items with PICTUREs from S9 through S9(4), in alphabetical order.

All other 77-level items should be grouped in alphabetical order.

All Ø1-level record descriptions should be listed in order of frequency of use.

• Liberal use should be made of spacing, page ejecting, and comments to make the WORKING-STORAGE coding more meaningful.

IBM GUIDELINES.

• The first 4,Ø96 characters of data in the WORKING-STORAGE SECTION are assigned a permanent base register. By putting the 77-level data items and the most referenced Ø1-level data item first, the most frequently used WORKING-STORAGE data may be referenced without resetting registers. This results in execution-time improvement.

• IBM COMPUTATIONAL items are aligned on boundaries according to the following schema:

Items described by PICTUREs corresponding to 1Ø through 18 decimal digits are aligned on doubleword boundaries.

Items described by PICTUREs corresponding to 5 through 9 decimal digits are aligned on fullword boundaries.

Items described by PICTUREs corresponding to 1 through 4 decimal digits are aligned on halfword boundaries.

The WORKING-STORAGE organization recommended will help make maximum use of main storage. The fields are arranged so that boundary alignment is automatic and the need for slack bytes between fields is eliminated.

## 2.7.1 DATA FORMAT CONSIDERATIONS.

### NUMERIC DATA ITEMS.

- GENERIC CONCEPTS.

The internal representation and use of numeric data items is a function of the compiler provided by each vendor.

- IBM GUIDELINES.

IBM supports the use of three data formats, two of which are permitted in SPEC coded modules: binary and external decimal. These are referred to in COBOL terminology as, respectively, COMPUTATIONAL and DISPLAY. For the purposes of the following discussions, DISPLAY refers to numeric DISPLAY items. The shortened form COMP for COMPUTATIONAL is used in this text.

The internal representation of a numeric data item is a function of the PICTURE and USAGE clauses pertaining to it. The following charts illustrate these relationships.

NUMERIC DISPLAY (External Decimal).

| PICTURE | USAGE | VALUE | INTERNAL REPRESENTATION |
|---|---|---|---|
| 9999 | DISPLAY | -1234 | F1 \| F2 \| F3 \| F4 |
| S9999 | DISPLAY | -1234 | F1 \| F2 \| F3 \| D4 |

Hexadecimal 'F' is treated arithmetically as a plus in the low-order byte. The hexadecimal character 'D' represents a negative sign. Therefore, it is important to apply a sign to the PICTURE clause for any numeric DISPLAY item which will be used in an arithmetic operation. If a sign is not used, invalid results may occur.

COMPUTATIONAL (Binary).

| PICTURE | USAGE | VALUE | INTERNAL REPRESENTATION |
|---|---|---|---|
| S9999 | COMP | 1234 | 0000 \| 0100 \| 1101 \| 0010 |
| S9999 | COMP | -1234 | 1111 \| 0100 \| 1101 \| 0010 |
| | | | (Sign Position) |

2.7.1 DATA FORMAT CONSIDERATIONS. (Cont.)

1. The allocated space for a binary item can contain a value much larger than the value implied by the PICTURE. For example, for a field defined with a PICTURE of S9(4), the maximum value is 9,999. However, the actual maximum value could be 32,767.

Generally, the programer does not need to be concerned with this situation. However, in the following cases, he must be very careful of the value of his data.

a. When the ON SIZE ERROR option is used, the size test is based on the implied maximum value as defined by the PICTURE clause of the result field. If a size error is detected, control passes to the imperative statements specified by the error option.

The programer must remember that the result field has not been altered and still contains the value which created the size error condition. This value is larger than that implied by the PICTURE clause of that field.

b. When a binary item is displayed or exhibited, the value used is a function of the number of 9s specified in the PICTURE clause.

c. Since a Ø-bit in the sign position means a number is positive and a 1-bit in the sign position means a number is negative, care must be taken not to let the value overflow into the sign position. For instance, when the actual value of a positive number is significantly larger than its picture value, a 1 could result in the sign position of the item, causing the item to be treated as a negative number.

2. The following table illustrates three binary manipulations. The result fields have been described as PICTURE S9 COMPUTATIONAL. The ON SIZE ERROR option was not used. If it had been specified, it would have executed for the last two items in the table.

| Hexadecimal Resultant of Binary Calculation | Decimal Equivalent | Actual Decimal Value in Halfword of Storage | Display Exhibit Value |
|---|---|---|---|
| ØØØ8 | 8 | +8 | 8 |
| ØØØA | 1Ø | +1Ø | Ø |
| C35Ø | 5ØØØØ | -15536 | 6 |

Effective use of the above data representation can be summarized in the following general rules.

1. Use 9's in a PICTURE only when arithmetic or editirg operations will be executed on the data item (for switches/flags use PIC X).

2.7.1 DATA FORMAT CONSIDERATIONS. (Cont.)

2. Unless the field is used as part of an edit PICTURE, the 9 should always be preceded by an operational sign, i.e., use PIC S9.

3. When a data item is used as a subscript or as the object in a GO TO ... DEPENDING ON, the usage of the data item should be COMP (signed binary).

DATA FORMAT CONVERSIONS.

- GENERIC CONCEPTS.

The data format selected can make considerable difference in the amount of object code generated and in the execution time of the operation. Data formats of a mixed mode require internal conversions to a common mode before execution takes place. This is especially true for mixed elementary numeric data formats. These items usually cause code to be generated which moves the item to an internal work area, converts it, and then executes the indicated operation. The result may also have to be converted.

The types of formats available and the conversions which take place are a function of each vendor.

- IBM GUIDELINES.

Program efficiency can be improved by minimizing the number of instructions generated to convert mixed mode data to a common format. The following techniques indicate ways this can be accomplished.

1. The same data formats should be used throughout the program for items which are frequently used together.

2. If it is impractical to use the same data formats throughout the program, a one-time conversion should be effected. The data should be moved to a work area which is in a format which does not require conversion for specified operations, (i.e., arithmetic operations). The work area can then be referenced whenever these operations are performed on that data item. This eliminates the need for internal conversions every time the item is used since the data is stored in the work area already in converted form.

3. Suggestions for working with numeric data fields.

a. Study the decimal requirements of your present file, then align the decimals of related fields on the converted data file.

b. Avoid mixed modes. Move frequently-used numeric DISPLAY fields to work fields defined as COMP to avoid multiple conversions.

c. Write literals with the same number of decimal positions as the receiving field.

2.7.1 DATA FORMAT CONSIDERATIONS. (Cont.)

d. Specify a sign with the picture, except when the sign is specifically unwanted.

e. COMP processing should only be used for whole numbers.

f. Literals should be used when possible.

2.7.2 DATA ITEM CONSIDERATIONS.

PREFIXING.

- GENERIC CONCEPTS.

Readability and transferability of programs are greatly enhanced if the data-names referenced are meaningful. The recommended technique for making these data-names more meaningful is as follows: (Note this technique is compatible with the standard names assigned in the USACSC Data Element Dictionary.)

File and Sort Description (FD/SD) Record Entries. Refer to the USACSC GUIDELINES under the FILE SECTION of this manual for USACSC prefixing rules.

WORKING-STORAGE SECTION entries. Refer to the USACSC GUIDELINES under the WORKING-STORAGE SECTION of this manual.

LINKAGE-SECTION entries. Refer to the USACSC GUIDELINES under the LINKAGE-SECTION of this manual.

The data element definitions (such as DATE) are defined in most cases in the file descriptions copied from the DATA ELEMENT DICTIONARY. These descriptions should be perpetuated with unique prefixes in those programer-defined records which are not copied (i.e., WORKING-STORAGE SECTION entries).

REDEFINES CLAUSE.

- GENERIC CONCEPTS.

The REDEFINES clause is an excellent COBOL tool for reducing main storage requirements. Two basic applications for the REDEFINES clause are the reuse of the same data area for different records and the redefinition of elements within a single record.

1. Reuse of data areas: The main storage area can be used more efficiently by writing different data descriptions for the same data area. The following example shows how the same work area can be used to define several different records which are not processed concurrently.

2.7.2 DATA ITEM CONSIDERATIONS. (Cont.)

```
WORKING-STORAGE SECTION.

Ø1  WS-WORK-FILE1.
    Ø3  WS-WORK-FILE1-DATA.
    .   (Largest file description for FILE1)
    .
    .
Ø1  WS-WORK-FILE2    REDEFINES WS-WORK-FILE1.
    Ø3  WS-WORK-FILE2-DATA.
    .   (Largest file description for FILE2)
    .
    .
```

2. Redefinitions of subfields: Program data can often be described more efficiently by providing alternate groupings of data descriptions for the same data. For example, a program may refer to both a field and its subfields each of which is more efficiently described with a different usage.

```
Ø1  MSTR-DATE1                  PIC S9(5).
Ø1  MSTR-DATE2E2            REDEFINES  MSTR-DATE1.
    Ø3  MSTR-YR-NUM             PIC S99.
    Ø3  MSTR-DAY-NUM            PIC S999.
Ø1  MSTR-DATE3              REDEFINES  MSTR-DATE1.
    Ø3  MSTR-YR-ALPHA           PIC XX.
    Ø3  MSTR-DAY-ALPHA          PIC XXX.
```

While the REDEFINES clause represents a useful tool, indiscriminate use of this clause should be avoided. Hierarchical, or nested, REDEFINES are often ambiguous and frequently misinterpreted.

For numeric data items, there is no savings in machine efficiency using any one technique. However, for better readability, literals used should correspond to the PICTURE of the data item being initialized.

```
    WS-FLD2                     PIC S9V99.

    MOVE +1.ØØ TO WS-FLD2.
```

2.7.2 DATA ITEM CONSIDERATIONS. (Cont.)

Group data items can also be initialized in both the DATA DIVISION and PROCEDURE DIVISION.

1. In the DATA DIVISION group items can be initialized either at the group level or each elementary item within the group may be initialized separately. By the use of the REDEFINES clause, the group item can be redefined as an elementary item. This is especially effective for initializing numeric fields and tables. For example:

```
WORKING-STORAGE SECTION.
     .
     .
     .
Ø1   WS-MESSAGE-TABLE.
     Ø3  FILLER                  PIC X(16)     VALUE     'ØØ1ADD MASTER..'.
     Ø3  FILLER                  PIC X(16)     VALUE     'ØØ2CHANGE MASTER'.
     Ø3  FILLER                  PIC X(16)     VALUE     'ØØ3DELETE MASTER'.
     .
     .
     .
     Ø3  FILLER                  PIC X(16)     VALUE     '/ /INVALID ADD '.
Ø1   WS-MESSAGE-TABLE-R REDEFINES WS-MESSAGE-TABLE.
     Ø3  WS-TAB-ITEM    OCCURS 2Ø TIMES.
         Ø5  WS-TAB-KEY          PIC XXX.
         Ø5  WS-TAB-IN-CLEAR     PIC X(13).
     .
     .
     .
PROCEDURE DIVISION.
     .
     .
     .
ØØ1Ø-AA-MATCH-MASTER.
     IF IN-TRANS-KEY IS EQUAL TO IN-MASTER-KEY
         MOVE 'Ø2Ø' TO WS-STORE-ERR
         GO TO ØØ2Ø-AZ-MATCH-EXIT.
     .
     .
     .
ØØ3Ø-WA-ERROR-REPORT.
     MOVE WS-TAB-IN-CLEAR (WS-STORE-ERR) TO WS-REP-ERR-LINE.
     .
     .
     .
```

2.7.2 DATA ITEM CONSIDERATIONS. (Cont.)

2. Data fields can be initialized in the PROCEDURE DIVISION using statements such as the following:

```
MOVE SPACES TO WS-WORK-DATA-2.

MOVE +99999 TO JUL-DATE-2.
```

Additional techniques for initializing tables are explored under TABLE HANDLING TECHNIQUES.

DECIMAL-POINT ALIGNMENT.

- GENERIC CONCEPTS.

PROCEDURE DIVISION operations are most efficient when the decimal positions of the data items involved are aligned. If they are not aligned, the compiler generates instructions to align the decimal positions before any operations involving the data items can be executed. This is referred to as *scaling*.

An example of inefficient coding due to decimal alignment:

```
WORKING-STORAGE SECTION.

77  WS-A            PIC S999V99.
77  WS-B            PIC S99V9.
    .
    .
    .
PROCEDURE DIVISION
    .
    .
    .
    ADD WS-A TO WS-B.
```

2.7.2 DATA ITEM CONSIDERATIONS. (Cont.)

This is inefficient because WS-B must be internally aligned so that it contains the same number of decimal positions as WS-A. This requires more compiler-generated coding and internal work areas.

Both time and internal storage space can be saved by defining WS-B as:

```
77  WS-B      PIC S99V99.
```

If it is not feasible to define WS-B in this manner, a one-time conversion might be done to an aligned work area in the program.

Literal values assigned to a data item should correspond in decimal alignment.

```
77  WS-C      PIC S9v99   VALUE 1.00.
```

FIELD INITIALIZATION.

- GENERIC CONCEPTS.

For readability, maintainability and as a debugging aid, data fields in a program should be initialized to a value prior to their use. Initialization of data fields is a function of both the data description and PROCEDURE DIVISION statements. The data item can be initialized using either the VALUE clause or by developing the value during the execution of the program.

2.7.2 DATA ITEM CONSIDERATIONS. (Cont.)

Elementary data items can be initialized using the following techniques:

1. In the DATA DIVISION, the VALUE clause using a literal or figurative constant can be used.

```
77  WS-A        PIC X      VALUE '   '.
77  WS-B        PIC X      VALUE SPACE.
77  WS-C        PIC S9     VALUE +1.
77  WS-D        PIC S9     VALUE ZERO.
```

2. In the PROCEDURE DIVISION, literals, figurative constants, or pre-initialized data names may be moved to a field to initialize it.

For alphanumeric items, the use of the literal or pre-initialized dataname is generally preferred to the use of a figurative constant. The use of a figurative constant causes extra coding to be generated to convert it to the size and format of the data item being initialized. However, if the item is exceptionally large, the use of the figurative constant may make the statement more readable and more maintainable.

```
77  WS-FLD1    PIC X(5).
77  WS-FLD2    PIC X(5)    VALUE SPACE.
```

2.7.2 DATA ITEM CONSIDERATIONS. (Cont.)

The following techniques are recommended for initializing FLD1.

```
MOVE '    ' to WS-FLD1.

MOVE WS-FLD2 TO WS-FLD1.
```

However, if WS-FLD1 was defined with a PICTURE X(500), it would be better to initialize it this way:

```
MOVE SPACES TO WS-FLD1.
```

EDITING.

- GENERIC CONCEPTS.

For efficient object program execution the following editing characters should be avoided, if at all possible:

1. DB.
2. + (plus sign).
3. - (floating minus sign).

When the currency sign ($) or check protection (*) symbols are specified, the rightmost $ or * should be preceded by the same character rather than by an insertion character. A PICTURE of $$$,999.99 is more efficient than $$$,$99.99.

A rightmost Z should not be preceded by a $.

BLANK WHEN ZERO should be avoided.

2.7.2 DATA ITEM CONSIDERATIONS. (Cont.)

The insertion character Ø should be used only in the leftmost character positions. Additional coding is generated for the execution of a MOVE statement when Ø is placed in other character positions. For example, a PICTURE of ØØØ,999.99 is more efficient than +++,ØØØ,999.99.

QUALIFICATION.

- GENERIC CONCEPTS.

Qualification of data-names is not allowed. The use of unique data-names is much more readable and less confusing. This makes program maintenance much easier. The careful use of prefixing makes qualification unnecessary. Like data items can be described the same way with only the prefix being different.

For example, these two descriptions:

```
JUL-DATE OF MASTER

JUL-DATE OF TRANS
```

can be just as easily and much more clearly defined as:

```
MSTR-JUL-DATE

TRANS-JUL-DATE
```

USE OF SIGNS.

- GENERIC CONCEPTS.

The absence or presence of a plus or minus sign in the description of an arithmetic field often can affect the efficiency of a program. The degree to which the program is affected depends on the internal representation of numeric data items by a particular vendor.

2.7.2 DATA ITEM CONSIDERATIONS. (Cont.)

A plus or minus sign: If 'S' is specified in the PICTURE clause, a plus or minus sign is inserted in one of two ways: the item is in WORKING-STORAGE and has had a VALUE clause specified or a value has been assigned as a result of PROCEDURE DIVISION operations (such as, arithmetic operations, MOVE instructions, etc.) during the execution of the program.

For numeric DISPLAY items, the presence of a plus or minus sign has special considerations if the field is punched, printed or displayed. If any of these operations occur, the sign digit is output with an overpunch in the low-order output digit. This results in representations:

1. Low-order digits of positive values will be represented by one of the letters 'A' through 'I'. These are the digits 1 through 9 with a hexadecimal 'C' in the zone portion of the low-order byte.

2. Low-order digits of negative values will be represented by one of the letters 'J' through 'R'. These are the digits 1 through 9 with a hexadecimal 'D' in the zone portion of the low-order byte.

3. Low-order zeroes in EBCDIC may be unprintable characters.

- IBM GUIDELINES.

For IBM's numeric configurations of internal or external decimal items, the sign position can contain a valid plus or minus sign, a hexadecimal F, or an invalid configuration.

1. A hexadecimal 'F': If 'S' is not specified in the PICTURE clause, a hexadecimal 'F' is inserted in the sign position in one of two ways: the item is in WORKING-STORAGE and has had a VALUE clause specified or a value has been developed during the execution of a program. The 'F' is treated as being positive but does not cause an overpunch when printed, punched or displayed.

2. An invalid configuration: If an internal or external decimal item contains an invalid configuration in the sign position, and the item is used in a PROCEDURE DIVISION operation, the program will abnormally terminate.

a. Unsigned items (no 'S' has been specified in the PICTURE clause) are treated as absolute values. Whenever a value (signed or unsigned) is stored in or moved in by an elementary move to an unsigned item, a hexadecimal 'F' is stored in the sign position of the unsigned item.

For example, an arithmetic operation involving signed operands and an unsigned result field will result in a hexadecimal 'F' being placed in the sign position of the result field.

2.7.2 DATA ITEM CONSIDERATIONS. (Cont.)

Note that extra code is always generated to erase the existing sign and place a hexadecimal 'F' in the sign field for unsigned items. Therefore, it is important to use an 'S' in the PICTURE clause for numeric items except when there is a vital well-defined reason not to do so.

b. For internal and external decimal items used as input, it is the user's responsibility to ensure that the input data is valid. The compiler does not test the sign position for validity.

c. When a group item is involved in a MOVE, the data is moved alphanumerically without regard to the level structure of the group items involved. It is possible for the sign position of a subordinate numeric item to be destroyed. Therefore, caution should be exercised in moves involving group items with subordinate numeric fields or with other group operations such as READ or ACCEPT.

DISPLAY ITEMS - NUMERIC VS ALPHANUMERIC.

- GENERIC CONCEPTS.

The relative value of defining DISPLAY fields in a program as numeric or alphanumeric is totally vendor-oriented.

- IBM GUIDELINES.

If a numeric DISPLAY field is not to be used for computations or in an operation requiring it to be in numeric format, it is more efficient to define this field as alphanumeric.

1. The use of a numeric PICTURE for DISPLAY fields for comparisons with literals, figurative constants or alphanumeric fields is very inefficient because of the extra coding that is generated to make a numeric comparison. Every time the field is interrogated, the data in that field is converted to packed decimal and a packed decimal comparison is made. This requires not only extra coding which results in increased execution time but also additional main storage for the internal work areas which are generated.

2. The use of an alphanumeric PICTURE produces a simple compare instruction with no intermediate work area.

These techniques are especially applicable to program flags.

2.7.3 PROCEDURE DIVISION TECHNIQUES.

2.7.3.1 Paragraph Naming. Paragraph naming conventions can increase the readability and intelligibility of programs; they can make it easier and faster to scan a program, to find paragraphs or sections. Subroutine paragraph names will be prefixed by a 4 digit numeric.

2.7.3.2 File Processing.

OPEN AND CLOSE ROUTINES.

- GENERIC CONCEPTS.

When multiple files are to be processed in a program, a determination must be made as to which savings is more important - processing time or main memory.

Each OPEN or CLOSE statement for a file requires the use of a storage area that is directly proportional to the number of files being opened or closed.

Opening or closing more than one file with the same statement is faster than using a separate statement for each file.

```
OPEN INPUT FILE1 FILE2 ...
```

Separate statements, however, require less storage.

```
OPEN INPUT FILE1.

OPEN INPUT FILE2.
```

Unless core storage is exceedingly limited, it is recommended that all opening and closing of files be done in parallel rather than separate statements.

When possible, all files should be opened at the same time and closed at the same time. This helps prevent abending by trying to open a file which is already opened or closing a file which is already closed.

2.7.3.2 File Processing. (Cont.)

READ AND WRITE STATEMENTS.

- GENERIC CONCEPTS.

READ and WRITE particularly need modular isolation. One READ or one WRITE per file is all that should occur in a program.

Any associated activity such as line controlling and page overflow should be logically located near the READ or WRITE.

REWRITE.

- GENERIC CONCEPTS.

This verb is only applicable for certain vendors.

- IBM GUIDELINES.

For mass storage files which require only minor updating the REWRITE statement is a useful technique.

The file must be opened in the I-O mode and a READ must be executed for each record before it is rewritten.

The use of this term requires only one storage allocation for the file. The file is read and written on the same physical location.

The REWRITE statement must be used with caution since the original record is not available after its execution. The user must provide some form of backup capability for files that are being rewritten.

2.7.3.3 Conditional Statements.

IF NUMERIC TESTS.

- GENERIC CONCEPTS.

When the IF statement is being used to determine if a field is NUMERIC and the last character of an otherwise numeric field contains a valid signed digit, the field is considered numeric.

- IBM GUIDELINES.

For the IBM system 360, valid sign is a hexadecomal 'C', 'D', or 'F'. However, the numeric test result depends upon the description of the data item being tested.

2.7.3.3 Conditional Statements. (Cont.)

1. If a data item is defined as a signed numeric field (S999), a valid sign can be a hexadecimal 'C', 'D' or 'F'.

2. If data item is defined as an unsigned numeric field (999), the only valid sign is a hexadecimal 'F'.

3. Note that it is important to be aware of the data description used when testing numeric fields.

IF STATEMENT OPTIONS.

- GENERIC CONCEPTS.

There are two options available for use of the IF STATEMENT - NEXT SENTENCE and ELSE.

The NEXT SENTENCE clause is usually redundant, generates unnecessary object coding and should be avoided. For example, the statement,

```
IS A IS EQUAL TO A NEXT SENTENCE
   ELSE ADD A TO B.
```

can be written more clearly as:

```
IF A IS NOT EQUAL TO B
   ADD A TO B.
```

This example results in main storage and execution time savings.

ELSE. When ELSE is used in an IF conditional statement, it should be positioned so that it is easily recognized. A suggested format is:

```
IF DATA-FIELD-A IS EQUAL TO DATA-FIELD-B

     PERFORM SUBROUTINE-A

ELSE

     PERFORM SUBROUTINE-B
```

2.7.3.3 Conditional Statements. (Cont.)

IF STATEMENT EFFICIENCY.

- GENERIC CONCEPTS.

Several abuses can arise in using the IF statement. These should be avoided:

1. Abbreviated forms of the IF statement (See example below).
2. Use of complex conditional statements.
3. Inefficient logic.
4. Nested IF statements with more than 3 levels.
5. Combinations of the above.

Abbreviated forms of the IF STATEMENT.

The COBOL structure can violate the normal, English-language logic of connectives. An IF statement in which the subject is implied may read:

```
IF T-TOTAL IS EQUAL TO Ø5 OR 1Ø OR ZEROS
    GO TO ØØ2Ø-PARA-B
```

The implied meaning is:

```
IF T-TOTAL IS EQUAL TO Ø5
    OR T-TOTAL IS EQUAL TO 1Ø
    OR T-TOTAL IS EQUAL TO ZEROS
    GO TO ØØ2Ø-PARA-B.
```

Confusion arises when a programer uses the following construct in which the subject and relational operand are implied:

```
IF T-TOTAL IS NOT EQUAL TO Ø5 OR 1Ø OR ZEROS
    GO TO ØØ2Ø-PARA-B.
```

2.7.3.3 Conditional Statements. (Cont.)

Although its intended meaning may be:

```
IF T-TOTAL IS NOT EQUAL TO Ø5
   AND T-TOTAL IS NOT EQUAL TO 1Ø
   AND T-TOTAL IS NOT EQUAL TO ZEROS
   GO TO ØØ2Ø-PARA-B.
```

Its actual meaning is:

```
IF T-TOTAL IS NOT EQUAL TO Ø5
   OR T-TOTAL IS EQUAL TO 1Ø
   OR T-TOTAL IS EQUAL TO ZEROS
   GO TO ØØ2Ø-PARA-B.
```

Obviously the programer won't be getting the object coding he expects. If confusion can arise from simple constructs, the confusion resulting from more complex constructs increases geometrically. Therefore, it pays off for programers to *spell out IF statements completely.*

Complex Conditional Statements.

Even when IF statements are fully written out, errors can result from using complex conditional statements. For example:

```
IF ONE-TOTAL IS EQUAL TO 1
   AND TWO-TOTAL IS EQUAL TO 1
   OR TWO-TOTAL IS EQUAL TO 2
   GO TO ØØ1Ø-PARA-A.
```

In order for the relational tests of TWO-TOTAL to be considered as a unit it must be parenthesized:

2.7.3.3 Conditional Statements. (Cont.)

```
IF ONE-TOTAL IS EQUAL TO 1
   AND (TWO-TOTAL IS EQUAL TO 1
   OR TWO-TOTAL IS EQUAL TO 2)
   GO TO ØØ1Ø-PARA-A.
```

Thus at the very least, such conditionals should be parenthesized and laid out for the best visual clarity by using the USACSC standard formatting as shown above. But obviously the best solution is to avoid such complex conditional IFs in favor of simple conditional statements.

Inefficient Logic.

1. The logic involved in an IF statement can sometimes be reversed and result in clearer coding.

```
     IF FLDA EQUAL TO 3
         GO TO ØØ4Ø-PARA-4
     ELSE
         ADD 1 TO FLDB.
ØØ4Ø-PARA-4.
```

The following example is clearer.

```
IF FLDA NOT EQUAL TO 3
   ADD 1 TO FLDB.
```

2. When testing for a value, the most likely occurrence should be tested first. The next most likely occurrence should be tested second. The next most likely should be tested next, etc.

Nested IF Statements.

Stringing IF statements together represents the most confusing and most difficult to maintain use of this statement. Because of this, USACSC Standards do not allow the use of more than 3 levels of nested IF statements.

2.7.3.4 Arithmetic Operations.

ADDITION AND SUBTRACTION.

- GENERIC CONCEPTS.

The most efficient ADD and SUBTRACT statements are:

```
ADD data-name-1 TO data-name-2
SUBTRACT data-name-1 FROM data-name-2
Where the decimal point scaling of data-name-1 is identical
to that of data-name-2.
```

Less efficient object coding is produced when the programer uses operands with different decimal point scaling with the GIVING, ROUNDED or ON SIZE ERROR option. However, this should not preclude the programer from using ON SIZE ERROR, ROUNDED and GIVING when he determines their necessity. Use of these terms may make the program more maintainable and this should be weighed before dismissing their use.

MULTIPLICATION.

- GENERIC CONCEPTS.

The multiplication of a numeric DISPLAY field by a power of 10 (.1, 10, 100, 1000, etc.) should be avoided. The same results can be obtained by redefining the field.

```
05  FIELDA                          PICTURE S9(5)V99.
     .
     .
     .
    MULTIPLY 100 BY FIELDA.
```

2.7.3.4 Arithmetic Operations. (Cont.)

The example on the previous page can be accomplished by redefinition which will result in a savings in main storage and execution time.

```
Ø5  FIELDB REDEFINES FIELDA  PICTURE S9(7).
```

ON SIZE ERROR.

- GENERIC CONCEPTS.

The ON SIZE ERROR is used with arithmetic operations. It occurs when the defined size of the result data field is smaller than the calculated result, or when the divisor is zero. The final result will not be accurate.

1. If the ON SIZE ERROR option is specified and a size error condition occurs, the resultant data-name is not altered and the series of imperative statements specified for the condition is executed.

2. The values involved in the arithmetic may be tested for limits if a size error condition is possible.

3. A test for a zero divisor should be made before dividing.

Size error conditions are possible when the operands involved in an arithmetic operation contain the maximum value allowable in a data description and the resultant field is not defined to contain the maximum allowable result.

- IBM GUIDELINES.

The ON SIZE ERROR condition applies only to final results. It does not apply to intermediate results.

INTERMEDIATE RESULTS.

- GENERIC CONCEPTS.

Compilers treat arithmetic statements as a series of operations and set up intermediate result fields to contain the results of these operations. The internal code generated and the size and description of these intermediate fields are different for each vendor.

- IBM GUIDELINES.

2.7.3.4 Arithmetic Operations. (Cont.)

The intermediate result fields generated by the IBM compiler depend upon the usage of the data items involved and the arithmetic statement being executed.

Binary Data Items.

If an operation with binary operands requires an intermediate result of greater than 18 digits, the compiler converts the operands to internal decimal before performing the operation. If the result is a binary field, the calculated result will be converted from internal decimal to binary.

If an intermediate result will not be greater than 9 digits, binary data fields are not converted. The operation is much more efficiently performed using binary data if no conversions are required.

Multiplication.

If a decimal multiplication operation requires an intermediate result greater than 30 digits, a COBOL library subroutine is used to perform the operation. The result is then truncated to 30 digits.

Division.

A COBOL library subroutine is used for division when one of the following occurs:

1. The scaled divisor is equal to or greater than 15 digits.

2. The internal decimal length of the scaled divisor plus the internal decimal length of the scaled dividend is greater than 16 bytes.

3. The scaled dividend is greater than 30 digits. (A scaled dividend is a number that has been multiplied by a power of ten in order to obtain the desired number of decimal places in the quotient.)

Intermediate Results Greater Than 30 Digits.

When the number of digits in an intermediate result is greater than 30, the field is truncated to 30 digits. A warning message will be generated by the compiler, but no abend will occur during execution. However, the truncation may cause an incorrect result.

ON SIZE ERROR.

This option applies only to the final calculated results and not to intermediate result fields. Problems can arise when intermediate results are not understood, as in the following example:

2.7.3.4 Arithmetic Operations. (Cont.)

```
COMPUTE  PERCNT = (A/B) * 1ØØ.
```

If A and B are defined as whole numbers, PERCNT will always be zero because the intermediate result will be truncated.

EFFECTS OF SIGNS.

- GENERIC CONCEPTS.

The use of the sign field (S) in the PICTURE clause for a numeric data item is important when arithmetic operations are performed using that field.

- IBM GUIDELINES.

If the result field of an arithmetic operation is not signed, extra instructions are generated to strip the sign from that field. Therefore, all result fields of arithmetic operations should be signed.

2.7.3.5 *Branching Statements*.

GO TO STATEMENT.

- GENERIC CONCEPTS.

The GO TO statement is considered one of the primary contributors to the complexity of computer programs. The GO TO when misused, creates "monolithic monsters" and therefore destroys the cleanliness of closed subroutine structures. It takes only a few intertwined branches in a program to exceed an optimum level of complexity.

Using PERFORM statements as a structure for subroutining and modularizing will significantly reduce the number of GO TO's in a program. However when the GO TO is used, there are guidelines which should be followed.

1. The object of the GO TO should only be a paragraph within the subroutine or module to which the GO TO belongs. Therefore, the programer does not have to look through the entire program to determine the impact of the branch.

This guideline also prohibits branching out of the middle of one subroutine into the middle of another one which destroys the independence of the paragraph-groups involved.

2.7.3.5 Branching Statements. (Cont.)

2. The GO TO should only branch forward in the program and only to an intermediate paragraph or EXIT paragraph within a subroutine or module. Branching backward implies a loop and the natural construct for a loop in COBOL is the PERFORM ... VARYING.

The ideal usage of GO TO is the branching forward only to the EXIT paragraph of a paragraph-group.

There are several operational questions the programer can ask himself as an aid in structuring a modularized, GO TO free program.

1. Is the mainline of the program concise and forward-moving?

2. Are the mainline paragraphs logically ordered?

3. Are any transactions bouncing through often repeated condition-testing?

4. Should the GO TO statement be replaced with a PERFORMed subroutine?

5. Are the subroutines properly modularized, that is, are they each performing one logical function?

GO TO ... DEPENDING ON ... STATEMENT.

- GENERIC CONCEPTS.

For less than 3 paragraph names, a comparison for equal conditions is just as efficient.

The GO TO ... DEPENDING ON should be well documented by comment entries to explain what the values of the object of the DEPENDING ON represent.

The programer must be careful when using this feature. Its use can be economical or costly in terms of memory and runtime. Following is an example of the DEPENDING ON feature used to best advantage.

1. Program A is a validation program accepting five types of transactions on a file. These transactions are identified by a three-position code. Program A assigns a number, 1 through 5, to a new field, TRANS-CD, which is included in the record written onto the output transaction file.

2. Program B is a master file update program which reads the transaction file, output of Program A. The processing of each transaction read into Program B can be determined as follows:

2.7.3.5 Branching Statements. (Cont.)

```
GO TO Ø11Ø-AA-ADD-MASTER
      Ø12Ø-AB-CHANGE-MASTER
      Ø13Ø-AC-INQUIRY-EXTRACT
      Ø14Ø-AD-DELETE-MASTER
      Ø15Ø-AE-CONTROL-CHANGE
      DEPENDING ON WS-TI-TRANS-CD.
```

IBM GUIDELINES.

The object of the DEPENDING ON should be defined with a computational PICTURE.

PERFORM STATEMENT.

- GENERIC CONCEPTS.

Most arguments used against use of the PERFORM verb say it is less efficient than a GO TO. Taken on a one-to-one basis PERFORM does require more machine instructions than a GO TO. What is often overlooked is the additional coding, such as IF statements and flag settings, it takes to implement more than one logical path passing through in-line coding.

A most important reason for using PERFORM is that it is one of the more powerful tools available in COBOL for modularity. A fuller treatment of the PERFORM statement and its use in modular program design can be found in PROGRAM DESIGN TECHNIQUES paragraph.

EXIT STATEMENT.

- GENERIC CONCEPTS.

The EXIT should always be used as the only sentence in the last paragraph of a subroutine. This aids program readability and self-documentation by making the subroutine limit clear to any user of the program.

The relationship of the EXIT verb with the PERFORM ... THRU is shown under LOOP CONTROLLING in PROGRAM DESIGN TECHNIQUES paragraph.

The return path of a PERFORM should always be through an EXIT.

2.7.3.6 Data Manipulation.

FIXED-LENGTH MOVE.

- GENERIC CONCEPTS.

The most efficient MOVE statement for unedited transfers, zero suppression and alphanumeric report editing is:

```
MOVE data-name-1 TO data-name-2.
```

For non-numeric elementary or group moves, the size of data-name-1 should be equal to or greater than the size of data-name-2.

A non-numeric literal may be moved more efficiently if the literal is made equal in size to the receiving field's size.

For a numeric-edited move, the scaling and computational size of data-name-1 should be identifiable with that of data-name-2. Computational size refers to the maximum number of numerics which can be present in the edited item. For example:

| EDITING PICTURE | COMPUTATIONAL SIZE |
|---|---|
| ZZZ.ZZ | 5 |
| $$$.99 | 4 |
| ---.99 | 4 |

2.7.4 TABLE HANDLING TECHNIQUES.

2.7.4.1 Table Construction and Referencing.

TABLE STRUCTURE.

- GENERIC CONCEPTS.

2.7.4.1 Table Construction and Referencing. (Cont.)

Tables are structured in the WORKING-STORAGE portion of the DATA DIVISION using the OCCURS clause. The organization of the table elements and their structure are interrelated with the types of techniques that will be used to interrogate the table.

```
Ø1  WS-TAB1.
    Ø3  WS-TAB1-ELMT        OCCURS      2Ø TIMES
                            INDEXED BY INDEX1
                                PIC X(1Ø).
```

- IBM GUIDELINES.

Large tables (greater than 4,Ø95 bytes) should be placed at the end of WORKING-STORAGE to avoid having assigned to them registers that would otherwise be assigned to more frequently accessed items (the 2nd thru nth register assigned to a large table is used only when executing index expressions composed entirely of literals).

TABLE REFERENCING.

- GENERIC CONCEPTS.

There are two techniques for referencing elements within a table: subscripting and indexing. Indexing is never slower than subscripting and may be up to 16 times faster.

Subscripting should be used as little as possible because of the complex conversions generated to convert the subscript (which represents an occurrence number in a table) to the address of the item being referenced. Therefore, subscripts should never by used when searching a table.

1. However, direct subscripting using literals or a data-field are useful. If only a direct reference need be made to a known occurrence number in the table, a literal subscript is efficient and makes the program more readable.

2.7.4.1 Table Construction and Referencing. (Cont.)

```
IF  WS-TAB-ELMT (1) EQUALS SPACES
                .
                .
                .
```

2. Direct subscripting using a data-name has limited use. However, if a data-field contains an occurrence number (such as a data-field in an input record) this could be used as a one-time reference. This should be used sparingly. If any computations need to be done for additional referencing, indexing should be used.

```
IF  WS-TAB-ELMT  (WS-INPUT-FIELD)  EQUALS WS-INPUT-KEY
                .
                .
                .
```

Indexing is faster than subscripting because an index is an internal binary counter which contains a displacement value which is added to the base address of the table to determine the address of the item referenced. Once an index has been set, it remains in a usable state without conversions being required. A subscript must be internally converted every time it is used even if the value of the subscript has not changed. Execution time, therefore, is saved when an index is used several times to refer to the same table element.

Execution time can be saved also by the use of efficient techniques for setting the value of the index. The order of preference for setting this value is:

1. SET TO another index-name which relates to a table having the same dimensions or to an index data item. Note that index data items are storage units for index-names. An index data item is described at the 77-level with the USAGE IS INDEX clause.

2. SET TO (UP BY or DOWN BY) a literal. Conversion instructions are not needed because the value of a literal is fixed at compilation time.

3. SET TO an index-name related to a table of different dimensions. Only a short version of the conversion procedure is executed at object time.

4. SET TO an identifier other than an index-name.

5. Examples of these techniques:

2.7.4.1 Table Construction and Referencing. (Cont.)

```
77  WS-DATA-ITEM                    PIC S9(5)  COMP.
77  WS-INDEX-DATA-ITEM              USAGE IS INDEX.
    .
    .
    .
01  WS-TAB1.
    03  WS-TAB1-ELMT                OCCURS 5 TIMES
                                     INDEXED BY WS-INDEX1
                                         PIC X(50).

01  WS-TAB2.
    03  WS-TAB2-ELMT                OCCURS 5 TIMES
                                     INDEXED BY WS-INDEX2
                                         PIC X(50).

01  WS-TAB3.
    03  WS-TAB3-ELMT                OCCURS 10 TIMES
                                     INDEXED BY WS-INDEX3
                                         PIC X(100).

PROCEDURE DIVISION.
0010-INIT.

    MOVE +1 TO WS-DATA-ITEM.
    SET WS-INDEX1 TO 1.
    SET WS-INDEX2 TO 1.
    SET WS-INDEX3 TO 1.
    .
    .
    .
0020-PROCESS.
    SET WS-INDEX-DATA-NAME TO WS-INDEX1.
    .
    .
    .
0030-EXAMPLE1.
    SET WS-INDEX1 TO WS-INDEX2.
    SET WS-INDEX1 TO WS-INDEX-DATA-NAME.

0040-EXAMPLE2.
    SET WS-INDEX1 TO 4.

0050-EXAMPLE3.
    SET WS-INDEX1 TO WS-INDEX3.

0060-EXAMPLE4.
    SET WS-INDEX1 TO WS-DATA-ITEM.
```

2.7.4.1 Table Construction and Referencing. (Cont.)

TABLE INITIALIZATION.

- GENERIC CONCEPTS.

All tables being constructed in a program should have their data areas initialized before construction. This aids in debugging and maintenance of the program. There are several techniques available for accomplishing this initialization.

In the DATA DIVISION, alphanumeric group items can be initialized using the VALUE clause at the group level.

```
01  WS-TAB1                                VALUE SPACES.
    03  WS-TAB1-ELMT                   OCCURS 10 TIMES
                                          PIC X(10).
```

In the PROCEDURE DIVISION, the technique used for initialization depends on the description of the table elements.

1. Alphanumeric table elements can be initialized simply by moving a value to the group level.

```
MOVE SPACES TO WS-TAB1.
```

2. Signed numeric table elements must be initialized individually in order to preserve the sign position. This can be done by indexing through the table.

```
01  WS-TAB1.
    03  WS-TAB1-ELMT                   OCCURS 10 TIMES
                                        INDEXED BY WS-INDEX1
                                          PIC S9(5).

PROCEDURE DIVISION.
0010-HSKP.
    SET WS-INDEX1 TO 0.
    .
    .
    .
0020-INIT.
    SET WS-INDEX1 UP BY 1.
    MOVE +0 TO WS-TAB1-ELMT (WS-INDEX1).
    IF WS-INDEX1 NOT EQUAL 10
        GO TO 0020-INIT.
```

2.7.4.1 Table Construction and Referencing. (Cont.)

- IBM GUIDELINES.

If a table has data elements which are COMP, the whole table can be initialized to ZERO by moving LOW-VALUES to the group level.

```
01  WS-TAB1.
    03  WS-TAB1-ELMT                    OCCURS 20 TIMES
                                             PIC S9(4) COMP.
    .
    .
    .
    MOVE LOW-VALUES TO WS-TAB1.
```

If a table contains numeric DISPLAY fields, each elementary item in order to be operated on arithmetically must have a sign associated with it.

The following method may be used to initialize the elementary items to ZERO.

```
01  WS-DATA-AREA.
    03  WS-DATA-DUMMY1.
        05  FILLER                          PIC S9(5)  DISPLAY VALUE 0.
        05  WS-DATA-DUMMY2.
            07  WS-DATA-VALUE           OCCURS 20 TIMES
                                             PIC S9(5) DISPLAY.
    .
    .
    .
    MOVE WS-DATA-DUMMY1 TO WS-DATA-DUMMY2.
```

2.7.4.1 Table Construction and Referencing. (Cont.)

A move from WS-DATA-DUMMY1 to WS-DATA-DUMMY2 propagates the value of WS-DATA-DUMMY1 throughout the area WS-DATA-DUMMY2 initializing each elementary item to ZERO with proper placement of the sign.

```
MOVE WS-DATA-DUMMY1 TO WS-DATA-DUMMY2.
```

This should be executed only when the area is to be initialized to ZERO.

2.7.5 TRANSFER OF CONTROL.

2.7.5.1 Overlay Structures.

GENERIC CONCEPTS.

- A diagram of a typical overlay structure is shown in FIGURE 2-65.

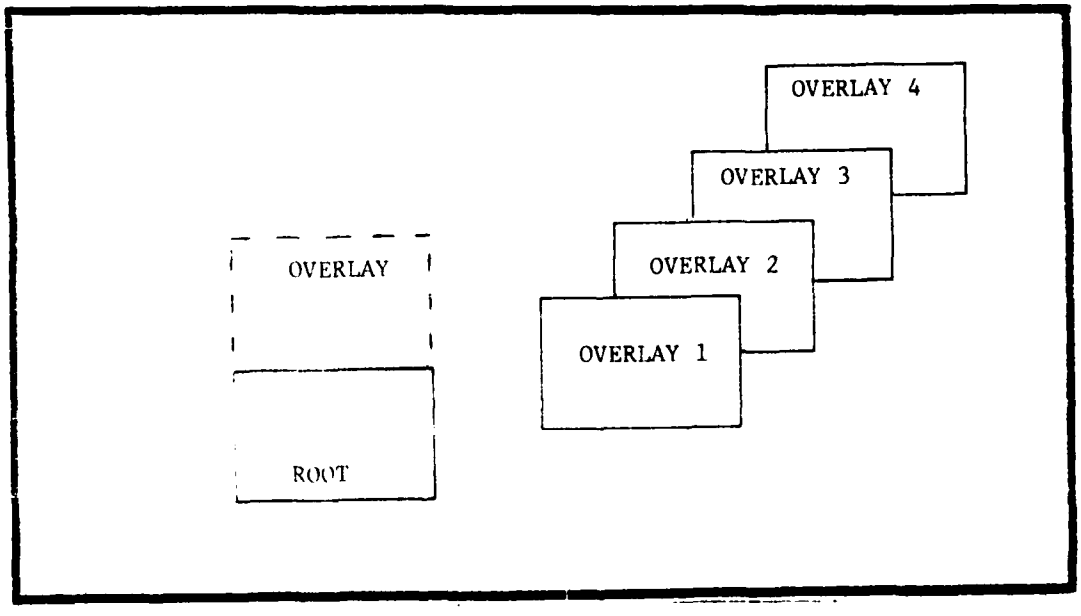



FIGURE 2-65

2.7.5.1 Overlay Structures. (Cont.)

The root portion remains core resident throughout the execution of the program. The root controls loading and execution of the overlays.

Overlay segments are loaded into core when CALLed by the root or other segments and can be overlaid by other segments.

An overlay module may be created through use of the COBOL segmentation facility. Refer to the segmentation feature in the Special Feature Section.

2.7.5.2 Subprogram Linkage.

GENERIC CONCEPTS.

- The CALL statement serves to pass control from one module to another module within a program. When control passes to a called module, execution is from procedure statement to procedure statement beginning with the first non-declarative statement. The logical end of the run unit is reached when a STOP RUN statement is executed and control is passed to the operating system. The logical end of the called module is reached only when an EXIT PROGRAM statement is executed; control then reverts to the next instruction following the CALL statement in the calling module. An EXIT PROGRAM in a non-subprogram is equivalent to a fall through.

- A called module can itself CALL other modules. However, where circularity of control is initiated an unusual end-of-program will occur. For example, if module A calls module B, then module B calls module A or another module which calls module A, an abend will result. The distinction between circularity of control and returning control to a calling module should be recognized. When A calls B and B calls C, control may return to B after the execution of EXIT PROGRAM in C and lastly control returns to A after the execution of EXIT PROGRAM in B. In this case, no circularity of control exists.

- Because a calling program may also be called, some confusion may arise as to whether to use the STOP RUN statement or the EXIT PROGRAM statement. Since a STOP RUN statement returns control to the operating system when encountered, it should not be used when the program is to be called. The EXIT PROGRAM, however, returns control to the program calling it and should not be used if the program is being used as the root calling program.

By specifying the following coding, both of these situations can be covered in the same program.

```
0090-EXIT-PROG.
    EXIT PROGRAM.

0100-STOP-PROG.
    STOP RUN.
```

2.7.5.2 Subprogram Linkage. (Cont.)

In this example, if the program is being called, the EXIT PROGRAM statement will return control back to the calling program. If the program has not been called, the EXIT PROGRAM statement will not be executed and control will pass to the STOP RUN statement which will terminate the execution of the run unit.

2.7.5.3 Subprogram Technique.

The technique to be used when calling COBOL subprograms is discussed in this section.

Reference should be made to using the PROCEDURE DIVISION USING ... option in the called program. This procedure is preferred to the ENTRY "entry point" USING ... technique which is not an ANSI standard but an IBM extension.

The LINKAGE SECTION provides useful core utilization. A called COBOL program, with the passed data defined within a LINKAGE SECTION, will reference the address locations defined in the main program. The called program will not establish duplicate allocations. The following is an example of a calling and called program with a LINKAGE SECTION. (Refer to FIGURE 2-66.)

CALLING PROGRAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  CALLPROG.
    .
    .
    .
DATA DIVISION.
    .
    .
    .
WORKING-STORAGE SECTION.
    .
    .
    .
Ø1  WS-RECORD-1.
    Ø3  WS-SALARY                    PIC S9(5)V99.
    Ø3  WS-RATE                      PIC S9V99.
    Ø3  WS-HOURS                     PIC S99V9.
    .
    .
    .
```

FIGURE 2-56

2.7.5.3 Subprogram Technique. (Cont.)

```
       PROCEDURE DIVSION.
       CALL SUBPROGRAM USING WS-RECORD-1.  STOP RUN.
           .
           .
           .
   CALLED PROGRAM

       IDENTIFICATION DIVISION.
       PROGRAM-ID.  SUBPROGRAM.
           .
           .
           .
       DATA DIVISION.
           .
           .
           .
       LINKAGE SECTION.
           .
           .
           .
       01  LS-PAYREC.
           03  LS-PAY                          PIC S9(5)V99.
           03  LS-HOURLY-RATE                  PIC S9V99.
           03  LS-HOURS                        PIC S99V9.
           .
           .
           .
       PROCEDURE DIVISION USING LS-PAYREC.
           .
           .
           .
       0090-END-SUB.
           EXIT PROGRAM.
```

FIGURE 2-66 (Cont.)

2.7.5.3 Subprogram Technique. (Cont.)

P52ATU, a USACSC OS executive software module, should be used as a link module whenever subprograms that are subject to a high degree of change and modification are used. The P52ATU module loads and transfers control to the subprogram at execution time. This allows changes to the subprogram without relinking the main program. The following techniques aid in clarifying which items in the called program are paired with items in the calling program.

All data items in the called program which are to be used by one or more calling programs should be physically grouped together.

The group of items should be combined into one Ø1-level record description. It is more efficient to reference one data-name with the USING clause than each individual item.

If the data items in a group cannot be combined into a single Ø1-level record description entry, they should be consolidated as much as possible and listed with the 77-level items first and the Ø1-level record description following.

The above techniques apply to the WORKING-STORAGE items for the calling program and the LINKAGE-SECTION items in the called program.

2.7.6 SOURCE LANGUAGE SYSTEM (SLS)/PROGRAM LANGUAGE UPDATE SERVICE (PLUS).

2.7.6.1 Source Library Maintenance.

• Source Language System (SLS) or Program Language Update Service (PLUS) will be used by all USACSC developers for source library maintenance. SLS will be used by all developers maintaining a single copy of a program coding that can be compiled and executed on different operating systems and/or across computer lines, for all 36Ø/37Ø OS systems, and for all developers using the 37Ø test bed. PLUS may be used for systems developed exclusively for IBM 36Ø DOS although the use of SLS is encouraged. PLUS may be used for systems converting from DOS to OS for a period of 12Ø days after completion of the conversion.

2.7.6.2 Catalogued Programs.

• All programs will be catalogued to the SLS or PLUS library prior to all updating or compilation.

2.8 SINGLE SOURCE LIBRARY SYSTEM.

2.8.1 OBJECTIVE.

• The objective of the single source library system is to establish a means of developing and maintaining a single copy of program source coding that can be compiled and executed on different operating systems and across computer lines.

2.8.2 PROCEDURES.

• All program source will be maintained using the USACSC standard source language maintenance systems. All unique coding that is needed to satisfy the requirements of various operating systems (OS vs DOS) or multi-vendor environment (IBM vs Burroughs, etc.), will be entered into the source language system. These lines of coding are identified as being unique to a specific requirement.

2.8.3 CODING.

• Column 7 of the COBOL coding format is used to indicate operating system or vendor unique coding. Current operating system codes are:

A in column 7 for OS only lines
B in column 7 for DOS only lines
C in column 7 for CDC only lines
D in column 7 for Debugging lines
E in column 7 for Burroughs only lines
F in column 7 for Honeywell only lines
G in column 7 for Series 1
H in column 7 for Univac only lines
* in column 7 for Comment lines
I-R reserved for future use

2.8.4 SINGLE SOURCE SYSTEM.

• Under the single source system, the source program is passed through a program which scans column 7 before the source is passed to the compiler. If the programer desires an OS compilation, this program would:

Upon encountering an "A" in column 7, replaces the "A" with a space activating that line for an OS compile and places an indicator in columns 73-80.

Upon encountering a "B", "C", "E", "F", "G", or "H" in column 7, replaces the character with an asterisk making that line a comment to the OS compiler and places an indicator in columns 73-80. The OS and all other unique lines remain OS comments.

For a DOS, CDC, UNIVAC, Burroughs, Honeywell or Series 1 compile, the opposite process would occur.

2.8.5 IMPLEMENTING INSTRUCTIONS.

• For OS users more detailed information including JCL required to use the Single Source Language System can be found in the USACSCM Executive Software Catalog, IBM/OS, Manual Number 18-2-B-ATU under program U05ATU.

## 2.9 OS/DOS COMPATIBILITY.

### 2.9.1 PROGRAM TECHNIQUES.

• Program conversions, from DOS to OS, have revealed several OS problems (causing abnormal program termination) that were found to be caused by COBOL programing techniques. These techniques have proven to be entirely satisfactory as far as ANSI COBOL syntax and DOS execution are concerned but have caused problems when operating under OS.

### 2.9.2 INPUT/STORAGE AREAS.

• DOS programers occasionally use input areas (the data areas associated with FD's or SD's) as storage areas. The use of these storage areas is often on a temporary basis as an intermediate work area. The DOS programer has found that it is more core efficient to use the input areas as compared to establishing similar areas in WORKING-STORAGE. DOS pointers to input areas are valid in circumstances where OS has random or incorrect pointer information. In OS, I-O buffer areas are available with these restrictions:

#### 2.9.2.1 Input Buffers.

• OS obtains input buffers when an OPEN statement is executed and releases these buffers when a CLOSE statement is executed. No moves to or from the data area of a closed file should occur. Such moves may cause abends or may overlay other data in the partition/region.

#### 2.9.2.2 Address Pointer.

• The OS address pointer to the input area is not established until after the first READ instruction. Again, a move to/from an OPEN but not read file gives unpredictable results. The output buffer area, however, is established when the output area is opened and data manipulations in the output area is permissible.

### 2.9.3 STOP RUN STATEMENT.

• The STOP RUN statement in DOS causes a return to the system and end-of-job step (EOJ macro). The STOP RUN statement under the OS system causes a return to the invoker of the main COBOL program as follows:

2.9.3 STOP RUN STATEMENT. (Cont.)

| INVOKER | RESULTS |
|---|---|
| . Operating System | . EOJ step |
| . Program in another language that follows COBOL invocation conventions. A program in another language that follows COBOL invocation conventions is logically considered to be a COBOL program. | . EOJ step |
| . Program in another language that does not follow COBOL invocation conventions. | . Return to the invoking program of the main COBOL program. |

A situation frequently encountered is the placement of a 'STOP RUN' statement within an INPUT or OUTPUT PROCEDURE. This will cause an ABEND 0C4. Likewise, a branch to a paragraph outside an INPUT or OUTPUT procedure causes an ABEND if a 'STOP RUN' is encountered.

2.9.4 DATA FORMATS.

• The DOS version of CURRENT-DATE has two formats, MM/DD/YY and DD/MM/YY. The OS format of these eight bytes is MM/DD/YY.

2.9.5 RECORD IDENTIFIER.

• Under DOS processing, there are no restrictions on the contents of the 'record-identifier' portion of the RELATIVE KEY. However, when processing F-mode records under OS, the first byte of the 'record-identifier' portion of the RELATIVE KEY must not be HIGH-VALUE else the system regards the record as a dummy record.

2.9.6 PROGRAM SWITCHES.

• OS cannot access the UPSI program switches. All references to these switches in the SPECIAL-NAMES paragraph and all switch conditions in IF, PERFORM and SEARCH statements should be removed.

2.9.7 PICTURE CLAUSE.

• The PICTURE clause for the RELATIVE KEY statement varies between DOS and OS. The PICTURE size is smaller in OS and the value of the track identifier is smaller. In addition, DOS allows the RELATIVE KEY to be specified under either actual track addressing or relative track addressing. OS allows only relative track addressing to define the RELATIVE KEY.

2.9.8 APPLY CLAUSE.

• Several options of the APPLY clause are normally handled in OS at execution time (JCL) whereas DOS would handle the techniques at compile time. The following are examples or clauses that must be removed from source coding and replaced by JCL.

(See IBM System 360/370 Operating System: Job Control Language Reference GC28-6704).

| APPLY CLAUSE | OS JCL (DCB PARAMETER) |
|---|---|
| APPLY EXTENDED-SEARCH | OPTCD = E<br>LIMIT = (No. of tracks) |
| APPLY WRITE-VERIFY | OPTCD = W<br>OPTCD = WC |
| APPLY CYL-OVERFLOW | OPTCD = Y<br>CYLOFL = (No. of tracks) |
| APPLY CYL-INDEX | OPTCD = M<br>NTM = (No. of tracks) |

2.9.9 INVALID KEY OPTION.

• Under DOS, for a randomly accessed file, the INVALID KEY option of the REWRITE statement is executed when the preceding READ statement caused an invalid key condition. When the OS READ statement for an indexed file causes an INVALID KEY condition, a REWRITE statement should not be executed for the record with that key since the INVALID KEY condition will not be detected and results are unpredictable.

2.9.10 SYNTAX ERRORS.

• Several additional differences exist however; syntax errors will be generated by the OS compilation. These differences are avoided by using USACSC COBOL which precludes the use of such special registers as COM-REG (Communication Region) and NSTD-REELS (Non-Standard Label Reel) count. The differences in the format of the system-name in the ASSIGN and RERUN clauses cannot be avoided, but must be recognized and accounted for since it is implementor defined by the COBOL language. A complete description of the system-name syntax and rules is in the section on USACSC Standard Coding Conventions.

# CHAPTER 3

# USACSC STRUCTURED PROGRAMING TECHNOLOGY

## 3.1 INTRODUCTION.

3.1.1 GENERAL. Computer programing has evolved over the years as an art rather than a science. Developments in recent years have demonstrated that, at least in some areas, a scientific or disciplined approach is possible.

Structured programing technology was introduced and implemented into USACSC by Plans 10-75 and 33-75. The Command has gained experience in structured programing technology primarily through the DS4, VIC, SAAS, VFDMIS, and IFS projects. The information acquired from these previous projects showed a need for a refinement of techniques.

The techniques which are included below are to be used for structured programing technology:

Top Down Development
Structure Chart
Data Flow Graph (Optional)
Program Design Language (PDL)
Programing Support Library (PSL)
Structured Coding (SC) or Structured Programing (SP)
Structured Walkthrough (SW) (Optional)
Team Operation or Chief Programer Team (CPT) (Optional)
Structured Testing
IPO (Input, Process, Output)
Nine Step Module Management Process (Optional)

The concept of Structured Coding followed, rather than preceded, the development of COBOL. In order to write structured code in COBOL, a programer must simulate the control logic structures or, with the aid of a preprocessor such as MetaCOBOL, utilize structure verbs.

3.1.2 PURPOSE. This chapter is to be used along with other chapters of this manual, particularly Chapter 2 which deals with USACSC standards for COBOL. It provides definition of terms, an introduction to the concepts, standards, and guidelines for implementation.

3.1.3 DEFINITIONS.

3.1.3.1 Backup Programer. In team operation, a backup programer is a senior programer and analyst who functions as an alternate to the chief programer so that he can assume the chief programer's responsibility temporarily or permanently.

3.1.3.2 Chief Programer. In team operation, a chief programer is a senior programer and analyst responsible for the complete development of the programing system.

3.1.3.3 Data Flow Graph. This is a graphic technique that highlights the data transforms in a program. A data flow graph will also show the conversion of input elements to output elements. It also helps define the structure of the program by showing the input, central transform (point where data is changed from input to output), and output legs of a program. A data flow graph can also be used as a review and documentation tool. Refer to Chapter 3 of TB 18-103 (Structured Design and Development) for an example of data flow graph.

3.1.3.4 IPO (Input, Process, Output) Chart - Shows the input, processes, and outputs for a program in a system. If a program consists of more than one module, an IPO chart can be drawn for each module and/or one can be drawn for the entire program.

3.1.3.5 Librarian. The librarian is a vital team member who transfers hard copy records into machine readable form. The librarian is depended upon for all assembly, compilation, linkage editing, and test runs as required by project programers. The results are filed by the librarian to maintain current status and history of the project.

3.1.3.6 Program Design Language (PDL) or PSEUDO-CODE. PDL is a near English like non-compilable language for describing the program logic. All the statements or instructions of this program are written in English like form, with indentation to show nested logic (loops). The actual source program can then be coded from this PDL, and it should generally be one statement of PDL generating one or two source statements. The term Systems Design Language (SDL) is also used sometimes, but it means the same as PDL or PSEUDO-CODE.

3.1.3.7 Programing Support Library (PSL). PSL is a repository for data necessary for the orderly development of computer programs using structured programing technology (SPT). The data repository is in two forms: internal library modules/segments are stored in machine readable form accessible by the computer and the external library identical data is stored in hard copy form in project notebooks. A PSL system also includes the necessary interlocking computer and office procedures for manipulating this data to provide a constantly up-to-date representation of the project and test data, together with an archive file for reference and backup purposes.

3.1.3.8 Structure Chart.

3.1.3.8.1 System Level. During the design state of a system, structure charts can be used to diagram the system. This diagram will show the top-level modules and each top-level module's subordinate modules. Also within each module is the title of that module.

3.1.3.8.2 Program Level. The structure chart depicts all the pieces (modules of a program and all the interconnections. It represents the actual physical structure of a program. (See FIGURE 3-2.) The structure chart is designed to show the physical structure of a modular program, hence, it does not directly show either the flow of data or control although explanation of the boxes in the structure chart is that a flowchart shows the flow of control, but does not show the data flow. IPO (Input/Process/Output) charts are tools used to further describe any given module in a structure chart. The IPO chart will divide the input, process, and output into columns and then list and describe the data elements for input, processing involved on data elements in the process column, and explain the output elements. A pictoral representation of IPO is shown in FIGURE 3-3 of this chapter.

3.1.3.9 Structured Program. Structured program is a program constructed of a basic set of control logic figures which provide at least the following: sequence of two operations, conditional branch to one of two operations and return, and repetition of an operation. A structured program has only one entry and one exit point. Logic flow always proceeds from beginning to end without arbitrary branching.

3.1.3.10 Structured Programing (SP) or Structured Coding (SC). To enhance the USACSC COBOL language and facilitate the implementation of structured programing, additional statements are available through the MetaCOBOL Macro Facility. These verbs should be used by the programer to accomplish structured module design. The structured verbs (DO, DO WHILE, IF, DO UNTIL, and CASE) are discussed in Chapter 2. SP is the process of developing the design, writing, and testing of a program which is made up of interdependent parts in a definite pattern of organization. Association with SP are certain practices and a set of rules added for clarity and readability, such as indentation of source code to represent logic levels, the use of meaningful data names, and descriptive commentary.

3.1.3.11 Structured Source Code Listing. Structured source code listing is a listing for a top down structured program (TDSP) comprised of the following sections:

3.1.3.11.1 The source list for the first executable structured module (commonly referred to as the top level module).

3.1.3.11.2 In alphabetical order, the source listings for all remaining structured modules.

3.1.3.11.3 Descriptive commentary (e.g., a table of inputs/outputs for each executable structured module).

3.1.3.11.4 The logical hierarchy of the structured modules which constitute this TDSP. It must show the executing sequence between executable structured modules.

3.1.3.12 Structured Testing.

3.1.3.12.1 Unit Testing. This type of testing is performed by using the same technique of top down development, whereby each program from the top down is tested as soon as it is written. Stubs may have to be developed to perform this, and they will be defined next. After it is tested, it is then implemented into the system to test lower level modules or programs and this is done down to the lowest level program. This testing is done by the programer.

3.1.3.12.2 Integration Testing. If a particular system can be broken into subsystems, then after all the unit testing of programs in this subsystem are written and tested, an integration test can be performed on it by the development group to see if this subsystem is performing its functions accurately and completely.

3.1.3.12.3 Systems Testing. This is performed by a quality assurance group (QAD) to develop an independent view of the results of the integration testing to see if all subsystems when integrated together will perform all functions specified. This quality assurance group will also check the completeness of the documentation.

3.1.3.13 Structured Walkthrough. In team operation, structured walkthrough is a generic name given to a series of reviews, each with different objectives and each occurring at a different time in the application development cycle. The reviewee subjects his work product to a review by other team members with the emphasis on design and coding error detection, thus contributing to increased team and project productivity.

3.1.3.14 Stub. This is a dummy module used to simulate an operational module. This is needed for top-down development technology whereby some upper level modules may need lower level modules for testing before this lower module is developed. In most cases, this stub will have only an entry and exit and possibly something to display that control was transferred to this module. When the lower level module is developed, this stub is discarded.

3.1.3.15 Team Operation or Chief Programer Team. Team operation is a group of programing specialists, consisting of chief programer, backup programer, and librarian, plus additional programers, analysts, and technicians required to complete the project.

3.1.3.16 Top Down Development. Top down development, is a method of program design and development. The end result is a TREE-LIKE hierarchical structure in which the highest level of control logic and decision is designed first and proceeds downward in increasing detail through lower level modules. The design phase is completed only when interface statements and initial data definitions are developed; at that point, top down coding may commence developing downward and top down testing may commence integrating the functional units in the same manner. Structure charts can be used at this level of the system (system level) to describe the system in a diagrammatic fashion. Within each diagram is the title of each module.

3.1.3.17 Top Down Program (TDP). In structured programing, a program can be one module or a group of modules. TDP is a structured program with the additional characteristics of the source code being logically, but not necessarily physically, segmented in a hierarchical manner and only dependent on code already written. Control of execution between segments is restricted to transfers between adjacent hierarchial segments. TDP is patterned after the natural approach to system design and requires that programing proceed from developing the control architecture (interface) statements and initial data definitions downward to developing and integrating the functional units.

3.1.3.18 Top Down Structured Programing (TDSP). TDSP is that part of the top down development process related to program coding and execution testing. With TDSP the top segment of a program and its required stubs are coded first and tested before proceeding in the development process. This process continues, one segment at a time from the top down, until program development is completed. Associated with TDSP are certain practices such as indentations of source code to represent logic levels, the use of intelligent data names, and descriptive commentary. TDSP requires TDP as the primary implementation methodology.

3.1.4 CONCEPTS OF TOP DOWN STRUCTURED PROGRAMING (TDSP).

3.1.4.1 General.

3.1.4.1.1 Functional Breakdown. TDSP starts with a desired function's specifications, repeatedly breaking down functions into simpler functions until reaching easily coded functions. TDSP, as applied to testing, is an ordering of system developing which allows for continual integration of the parts being developed. At each state, the code already tested drives the new code, and only external data requires no extra drivers.

3.1.4.1.2 Modular Structure. In top down programing, the system is organized into a network structure of modules. The top module contains the highest level of control logic and decisions within the program, and either passes control to next level modules, or identifies next level modules for in-line inclusions. The incomplete next level modules are called "stubs" and those which are to be replaced eventually with running code may contain a "no operation" instruction or possibly a display statement to the effect that control had been received. While it is recognized that such code, as with drivers, are also eventually discarded, the effort involved in writing such statements is less than that required to produce and pass data to a module for unit testing. The process of replacement of successively lower level stubs with processing code continues for as many levels as required until all functions within a system are defined in executable code.

3.1.4.1.3 Data Base Definition. Top down programing requires that sufficient data base definition statements be coded and that data records be generated before exercising any module which references them. Ideally, this leads to a single set of definitions serving all the programs in a given application. This approach provides the ability to evolve the product in a manner that maintains

3.1.4.1.3 Data Base Definition. (Cont.)

the characteristic of being always operable, extremely modular, and always available for successive levels of implementation. The quality of a system produced using this approach is increased, as reflected in fewer errors in the coding process. The act of structuring the logic and structured design techniques calls for more forethought, and the uniformity and single entry, single exit attribute of the structured code itself contributes to the reduction in errors. Because of the nature of TDSP, the resulting system is extremely modular in function and logic structure, minimizing the effect of requirement changes on already-developed code.

3.1.4.1.4 Top Down Development. Conceptually, TDSP proceeds from a single starting point while conventional implementation proceeds from as many starting points as modules in the design. The single starting point does not imply that the implementation must proceed down the hierarchy in parallel. Some branches intentionally will be developed earlier than other branches. For example, user or other external interfaces might be developed to permit early training or hardware/software integration. Also in many applications, requirements will become firm in certain areas before others. The areas covered by known requirements can usually become operational while the requirements are being developed for the others. Some module stubs intended to support long-range requirements may remain after the program is fully operational as a guide if and when they are needed.

3.1.4.2 Figures.

3.1.4.2.1 FIGURE 3-1 shows a comparison of traditional bottom up development and top down development. (The dash lines represent the unfinished portions of the programs.)

3.1.4.2.2 FIGURE 3-2 is an example of a structure chart at the program level.

3.1.4.2.3 FIGURE 3-3 is an IPO example of an overview diagram of the Gross Pay Calculation Module and its subordinate modules. There can be a detailed IPO for each module all the way down to the lowest level module if desired.

3.1.4.3 Program Design Language (PDL) or PSEUDO-CODE. This paragraph prescribes guidelines to be observed in the use of a Program Design Language (PDL). PDL along with structure charts, is a textual alternative to flowcharts and decision tables, and is particularly well suited for use in conjunction with SP. A PDL module consists of a logically complete group of PDL statements, and is used to state the program logic designed to satisfy a detailed functional requirement. PDL, due to its textual form, is more easily maintained than graphic forms of presentation.

3.1.4.3.1 Purpose. The purpose of a PDL is to provide a vehicle to translate functional specifications into program design, based on TDSP. It is used to convey logic design when the coded solution is not readily apparent from the specification.

3.1.4.3.2 Description. PDL is a formatted English textual presentation of a program logic that is prepared by the analyst or programer/analyst. This PDL is prepared for use in a code walkthrough. It can also be used in communicating with other people that may not know the computer program language (i.e., COBOL). After the code walkthrough is completed and determined to be correct, the actual source program is coded from this PDL by the programer. The grammatical structures prescribed parallel those required for structured coding, i.e., the basic control structures. Indentation rules are defined to increase readability. Selected keywords, used to outline the structure of PDL, are reserved to enhance communication among users, but the majority of PDL vocabulary is left to the choice of the system developers. In the following subsections, keywords are written in capital letters; brackets ([ ]) and dotted lines indicate optional items.

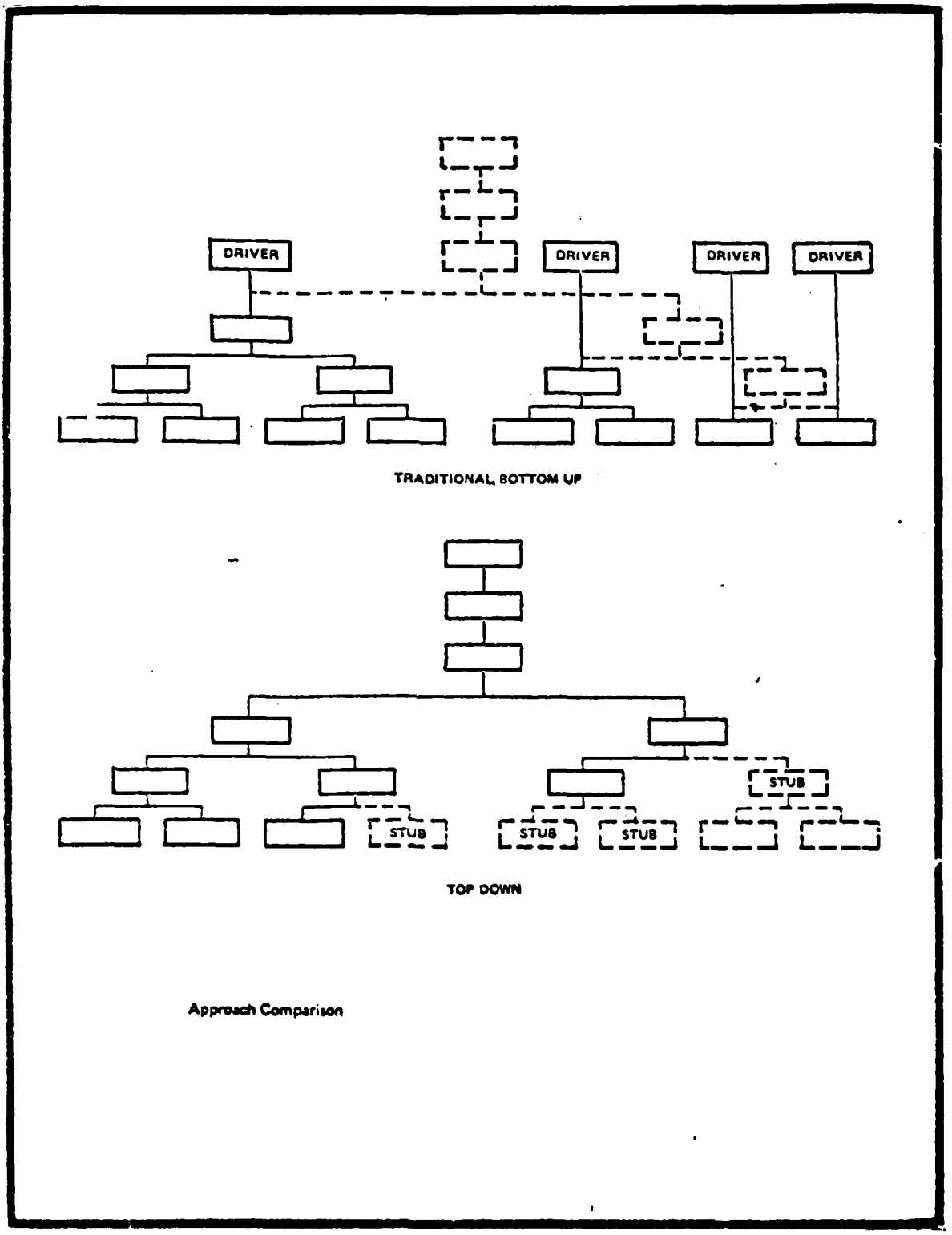

DRIVER
DRIVER
DRIVER
DRIVER
TRADITIONAL BOTTOM UP
STUB
STUB
STUB
STUB
TOP DOWN
Approach Comparison

FIGURE 3-1

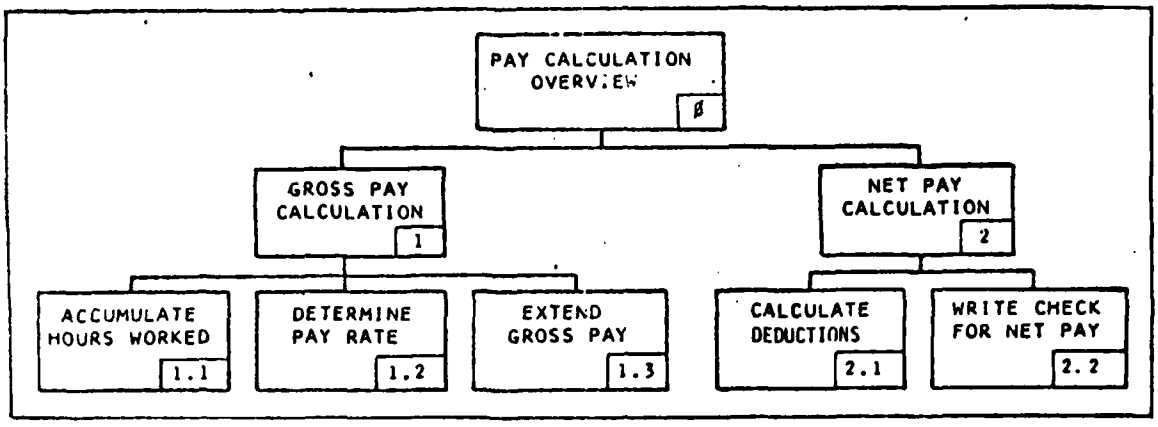



FIGURE 3-2

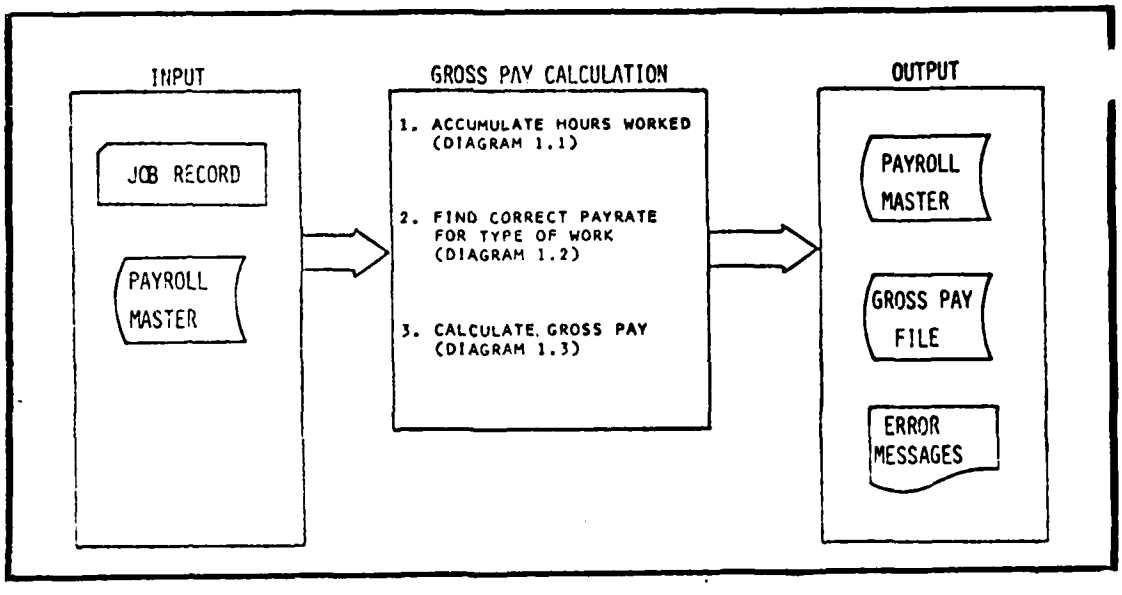



FIGURE 3-3

3.1.4.3.3 Module Delimitation. The beginning and end of modules are identified, respectively, by a user-selected module name and the word 'end' both beginning in the leftmost position of the PDL text. A key word identifying the module type may follow the module name. The portion of the PDL which related the action of the module begins at a 2-columnm indentation. The following illustrates overall module structure.

```
Module name [segment type]
  module action Test
END Module name
```

3.1.4.3.4 Types of PDL Segments (Modules).

3.1.4.3.4.1 Main - represents the top-level control of the program. It includes the program start and stop operations.

3.1.4.3.4.2 Subroutine - represents a discrete section of executable code performing a complete operation. It is physically separate from the program and will be called for at time of execution.

3.1.4.3.4.3 Included - represents executable code physically included in the program at the time of compilation.

3.1.4.3.4.4 Data - describes a section of program which only contains data definitions.

3.1.4.3.5 Internal Module Structure - PDL text describing action will be indented 2 columns from the module name. PDL keywords which outline specific logical figures will start at this margin of indentation. When figures are nested, indentation is cumulative.

3.1.4.3.6 Guidelines for Semantics. The choice of vocabulary to be used to describe predicates and functions is left to the development group. Selections may be standard English language and/or may be derived from the intended implementation language. Most choices should be made on a projected basis, not varying between programers within a project. The goal of the PDL is maximum comprehensibility when transferring technical information among diverse groups of managers, programers, and analysts.

In addition to the keywords already described, two others are reserved for use in the action portion of a PDL module.

INCLUDE - This preceeds the name of a module to be included (INCLUDED or DATA type module).

CALL - This preceeds the name of a subroutine to be executed (SUBROUTINE type module).

3.1.4.3.7 USACSC Program Design Language (PDL) Conventions. The Program Design Language (PDL) will be subject to the following rules:

3.1.4.3.7.1 Only the five logic structures (sequence, if-then-else, do while, do until, and case) will then be used.

3.1.4.3.7.2 Indentation will be used. It is recommended that indentation when used be 2 for dependency or 4 columns for continuation.

3.1.4.3.7.3 There will be only one logical statement per line.

3.1.4.3.8 PDL Example. Below is an example of PDL representation.

```
MØØØSOT WAFRE MAIN

   DO INITIALIZATION
   DO MØØ1SOT-OBTAIN CLEAN WORD
   DOWHILE (MORE WORDS)
     DO MØØ2SOT-UPDATE TABLE
     DO MØØ1SOT-OBTAIN CLEAN WORD
   ENDO
   DO MØØ2SOT-DUMP TABLE
   DO TERMINATION
END MØØSOT

MOO2SOT UPDATE THE TABLE INCLUDED
   CALL "MODIFY"
   IF (TABLE-FULL)
     DO MØØ3SOT-DUMP WORD FREQUENCY TABLE
     CALL "MODIFY"
   (ELSE)
   ENDIF
END MØØ2SOT

MØØ3SOT DUMP WORD FREQUENCY TABLE INCLUDED
   SET NEXT-WORD-POINTER TO 1
   DOWHILE (NOT END-OF-TABLE)
     CALL "DUMP"
     DO MØØ6SOT-FORMAT LINE
     DO DFO2SOT-PRINT LINE
     INCREMENT NEXT-WORD-POINTER BY 1
   ENDDO

END MØØ3SOT
```

3.1.4.4 Programing Support Library (PSL).

3.1.4.4.1 General.

3.1.4.4.1.1 A Programing Support Library (PSL) serves as a repository for data necessary for the orderly development of computer programs using SPT. The data exists in two forms.

INTERNAL LIBRARY. Data is stored in machine readable form accessible by the computer.

EXTERNAL LIBRARY. Corresponding data is stored in hard copy (Human readable) form in project notebooks.

Included with PSL are the necessary computer and office procedures for manipulating this data.

3.1.4.4.1.2 The purpose of a PSL is to support the program development process. This involves the support of the actual programing process and the management of the process. Support of the programing process involves support of the design, coding, testing, documentation, and maintenance of computer programs and the associated data-base definitions. A PSL provides this support through:

Storage and maintenance of programing data (Segments/modules).
Output of programing data and related control data.
Support of compilation and testing of programs.
Support of the generation of program documentation.

3.1.4.4.1.3 A PSL must also provide some means of generating and maintaining itself. Support of the management of the programing process also involves the storage and output of programing data. In addition, it involves:

Collection and reporting of management data related to program development.
Control over the integrity and security of the data stored in the PSL.
Separation of the clerical activity related to the programing process.

3.1.4.4.1.4 A PSL supports the approach in which people work on a common, visible, centrally-stored product rather than on independent hidden pieces. It provides a significant aid for test and evaluation in that the current operational software system code is centralized to avoid ambiguity of what is, and what is not, valid software as well as centralizing the valid test program code. The programers communicate through this product in carrying out programer and clerical interface activities. A PSL permits a programer to exercise a wider span of detailed control and reduces explicit communication requirements. This makes it easier to bring new personnel on board and to shift programers from one part of the project to another. It also minimizes the preparation effort for technical audits. A librarian is responsible for maintaining the notebooks and archives of the PSL, and the programers are responsible for their source code. This structure of responsibility permits standardization in project record keeping and insures that the hard copy listings in the library correspond to the most current version of the system. A PSL system has four components:

Internal Libraries
External Libraries
Computer procedures
Office procedures

The components of the system are interlocked to establish an exact correspondence between the internal (Computer readable) and external (Programer readable) versions of the developing system. This continuous correspondence is the characteristic of a PSL that guarantees on-going visibility of the developing system.

3.1.4.4.1.5 Different implementations of a PSL exist for various computer and operating system environments used in system development. The fundamental correspondence between the internal and external libraries in each environment is established by the PSL office and computer procedures. The office procedures are specified at a detailed level so that the format of the external libraries will be standard across programing projects, and so that the maintenance of both internal and external libraries can be accomplished as a clerical function. The PSL computer procedures for each are expressly designed for easy invocation by librarians so that their use is nearly fail-safe.

3.1.4.5 Structured Walkthroughs. Project management has long recognized the need for periodic reviews to determine whether the project is on schedule and to identify areas that require special attention. Generally, however, these exercises have been looked upon with hostility by those who must submit themselves to the review. A structured walkthrough is a generic name given to a series of reviews; each with different, but specific objectives and each occurring at different times in the software development cycle. The basic characteristics of the walkthrough are:

3.1.4.5.1 It is arranged and scheduled by the developer (person responsible for the work product being reviewed).

3.1.4.5.2 Participants (reviewers) are given the review materials prior to the walkthrough and are expected to be familiar with them.

3.1.4.5.3 Management above team leader (chief programer) will not be involved in structured walkthrough(s) and structured walkthroughs will not be used for employee evaluation.

3.1.4.5.4 The emphasis is on error detection rather than error correction.

3.1.4.5.5 Typical walkthrough will include four to six people and will last for a specified time, usually one or two hours. If at the end of this time the objectives have not been met, another walkthrough is scheduled.

3.1.4.5.6 All technical members of the project team have their work product reviewed. The structured walkthrough increases the value of these reviews beyond a determination of schedule variance and problem identification, and eliminates many negative aspects.

3.1.4.5.7 Specification Walkthrough - This type of walkthrough is done in the first stages of the system. Its purpose is to look for problems or omissions in the system specifications. This type of walkthrough would be attended by the user, systems analyst, and one or more of the project programers.

3.1.4.5.7.2 Design Walkthrough - This walkthrough would be used to detect weaknesses, misrepresentation of ideas, or omissions in the design structures of the system. The people that should attend this walkthrough are the system analyst, senior programer (Chief Programer) and possibly all the other programers as well. The documents used in this walkthrough would be the structure charts and data flow graphs.

3.1.4.5.7.3 Code Walkthrough - In this walkthrough, the two types of code are reviewed. These two types of code are the Program Design Language (PDL) or Pseudo Code that is designed by the analyst in an English like form, and after this PDL is walked through and certified as correct by people other than the author, then the programer prepares the source code from this PDL. Then, this source code is reviewed and certified as correct by people other than the author of the source code. The certification should be performed by people other than the author for every code walkthrough. The people that should attend this walkthrough are the author of the code, other programers on the team, and possibly, some programers from outside the team.

3.1.4.5.7.4 Test Walkthrough - This walkthrough is performed after the source code has been certified. It is performed to ensure the adequacy of the test data for the system. This walkthrough does not examine the output of the test data. People that should attend this walkthrough are the user, all the programers on the project, the person performing the testing and the systems analyst.

3.1.4.5.8 In addition to all the various groups of people stated in these four types of walkthroughs, the librarian should be present at most walkthroughs, depending on his/her workload. It is the librarian's job to record items in question, the results, record items accomplished and at the conclusion of each walkthrough, distribute a copy of all items from above to all members that were present at that specific walkthrough. If the librarian is not able to attend a walkthrough, someone else in the group should be appointed to act as librarian.

3.1.4.6 Chief Programer Team (CPT). The team operation has as its core, the chief programer, the backup programer, and the librarian. The chief programer is responsible for the complete development of the programing system. The chief programer identifies and apportions assignments, constructively criticizes progress on design and coding, attends to career planning for his subordinates, and regularly performs the appraisal and counseling of team operation. The backup programer is an alternate to the chief programer. He can assume the chief

3.1.4.6 Chief Programer Team (CPT). (Cont.)

programer's responsibility temporarily or permanently. Because of the close working relationship and code review practices, both understand all the code produced by the team.

3.1.4.6.1 Librarian. A programer technician or specially trained secretary that is an integral member of the Chief Programer Team. The librarian's duties are largely clerical that consist of the following: key punching, submitting jobs, maintaining program production library, and taking formal notes at structured walkthroughs.

3.1.4.6.2 Support Members. The team operation may require additional support to complete the project. People who serve on the team - programers, analysts, and technicians - are chosen for their special skills. Their period of service may range from a few months to a period just under the length of the project.

3.1.4.7 Nine Step Module Management Process. The nine step module management process developed and refined by the VIC project provides a sound basis for the determination of software development status. This procedure should be considered for software production and control procedures. The steps are as follows:

3.1.4.7.1 Step 1 Module Identified. The module is identified as a firm requirement, its position on the total system structure chart relative to superordinate and subordinate module is determined, and a module number is assigned.

3.1.4.7.2 Step 2 Module Documented. The systems analysis and design have been completed, and a package of documentation for the module has been prepared for presentation to the functional proponent.

3.1.4.7.3 Step 3 Joint Review of Module. A joint review of the module has been conducted in a walkthrough environment with the functional proponent for functional modules, or the design manager for data modules, and an action list detailing required changes has been prepared.

3.1.4.7.4 Step 4 Module Accepted. Any corrections required by the joint review have been made, the module package is considered complete, has been formally accepted, and is ready for programing.

3.1.4.7.5 Step 5 Module Coded and Compiled. The module extended description has been jointly reviewed by the chief programer and backup to ensure complete understanding of the requirements. The module has been coded and cleanly compiled and it is considered ready for quality assurance review.

3.1.4.7.6 Step 6 Module Test Plan Complete. A test plan has been completed for the major function of which this module is part, incorporating all of the conditions (TCR's) necessary to test this module, showing the sequence of testing for each condition, and the expected output for each condition.

3.1.4.7.7 Step 7 Module Quality Assurance Review. A quality control review has been held to ensure that the compiled code complies with all necessary USACSC standards, in-house conventions, structured coding conventions, and that the code should perform the functions described in the approved extended description. In addition, the test plan has been reviewed and approved as being comprehensive enough to adequately satisfy testing requirements.

3.1.4.7.8 Step 8 Module Linked. All corrections required by the quality control review have been applied, the module has been cleanly compiled and is linked to the library and is ready for testing. The test data (TCR's) are available and testing can begin.

3.1.4.7.9 Step 9 Module Ready. All processing functions/conditions performed/encountered by the module have been tested to the satisfaction of the software development team and the module is considered to be ready for full scale systems testing utilizing proponent developed test data.

3.1.5 GENERAL STANDARDS AND GUIDELINES. There are two types of conventions followed when writing structured programs - standards and guidelines. Standards are those conventions which are to be followed without deviation. However, a waiver is available to consider justifiable requests for deviations from the standard. Guidelines are recommendations which can be followed exactly or with appropriate local deviation. Thus, standards to achieve maximum benefit should be enforced throughout an organization. Guidelines should be modified as appropriate for a particular project or system development. The use of a specific set of control structures is a standard. The indentation of source code is a standard; the number of columns indented is a guideline.

3.1.5.1 Standards. Certain standards must be followed in order to implement SP. These are:

3.1.5.1.1 Every code segment should contain a single entry and a single exit.

3.1.5.1.2 Explicit branching (GOTO type instructions) usage is restricted to emergency type exits, i.e., specific compiler bugs or inefficiencies, data error, and/or panic abort.

3.1.5.1.3 The beginning and end of any program or control segment must be completely contained in a single structured segment.

3.1.5.1.4 In free format languages, only one verb per line of code is permitted.

3.1.5.1.5 Indentation to indicate the span of control of a structured figure must be used.

These standards are language independent. They are based directly on the structure theorem, or on the intent to make computer programs as readable as possible.

The USACSC SPEC COBOL standards which are indicated in this chapter are intended to serve as a base upon which production structured programing can be implemented. They should be evaluated continuously, and as experience dictates, improvements should be incorporated into the standards.

3.1.5.2 Guidelines. In addition to specifying the USACSC SPEC COBOL language dependent standards for implementing the control logic structures, this section also contains other recommendations which may be considered as guidelines. This covers such items as indentation rules, grouping of data, data formats, etc. The most important consideration with respect to guidelines is not that the ones described be implemented exactly as indicated but rather that, for a given project, conventions be established for the indicated areas, and then applied uniformly throughout the entire project.

3.1.6 USACSC SPEC COBOL LANGUAGE STANDARDS AND GUIDELINES. Refer to Chapter 2 for the format and guidelines for SP verbs (DO, DO WHILE, IF, DO UNTIL, and CASE).

3.1.7 INCLUDE CAPABILITY. The capability of nesting blocks of code within other code blocks is a necessity for top down programing. This is best done when such blocks of code are stored and can be accessed by the ANSI COBOL COPY verb as separate members on a direct access device in a library system. However, it should be noted that this requirement is a compiler dependency and may not be possible for some ANSI compilers. The COPY verb may be used in all divisions except the IDENTIFICATION DIVISION. The discussion which follows is directed towards its use in the PROCEDURE DIVISION. Since the COPY does not permit nesting, it is necessary to simulate this requirement with the use of nested PERFORM statements. The blocks of code which are PERFORMed are presumably stored as separate members which are easily accessed on a direct access device and are referenced for the COBOL compiler by means of COPY statements. This means that no COPY may appear in any block of code which is invoked by a copied PERFORM. With this technique the top level of the PROCEDURE DIVISION looks as follows:

```
PROCEDURE DIVISION.
TOP-PARAGRAPH.
    CODE A.
    PERFORM NESTED-PARAGRAPH-1.
    CODE B.
    STOP RUN.
NESTED-PARAGRAPH-1.
    COPY LIBRARY-NAME-1.
NESTED-PARAGRAPH-2.
    COPY LIBRARY-NAME-2.
```

3.1.7 INCLUDE CAPABILITY. (Cont.)

Note that the COPY statement references library names, not paragraph names. "NESTED-PARAGRAPH-1" is a separate block of code which the COPY statement can access and may take the following form:

```
CODE C.
PERFORM NESTED-PARAGRAPH-2.
CODE D.
```

"NESTED-PARAGRAPH-2" is a sequence of statements similar to those contained in the above paragraph within which it was invoked and it may contain other PERFORMs for deeper nesting. The COPY statements following the top paragraph ensure that the compiler is aware of all the segments of code which comprise the total program. Furthermore, since no PERFORMed paragraph may contain a COPY, there is no danger of violating the nesting limitation of this verb.

3.1.7.1 Additional Recommended Coding Conventions.

3.1.7.1.1 Restricted COBOL Statement Usage. In order to preserve the concept of SP, it is recommended that the general usage of these statements in COBOL which permit changes of sequential control be restricted to an exception basis only. The ALTER statement will not be used.

3.1.7.1.2 Program Organization.

3.1.7.1.2.1 The structure of a COBOL program is such that many of the rules for program organization have been predefined. For instance, all data must be specified in the DATA DIVISION. Furthermore, within this section, the formal rules which define the permissible hierarchical data structures are sufficient to preserve the readability requirements of SP. However, within the PROCEDURE DIVISION (with the exception of the DECLARATIVE SECTION), the rules of COBOL permit the ordering of PERFORMed code blocks to be completely flexible.

3.1.7.1.2.2 If the program is being developed with the aid of a library system, the order in this division is less critical since all that appears after the top most segment are COPY statements. The functions which exist in the copied code and the functions which are nested within them are determined by examining the small code segments which are present as printed listings of members in the source code library rather than on the compiler output listing even though it is still true that the resolution of the COPY statements by the compiler will produce a complete source program as one of the compiler outputs. However,

for a development process in which no random access library exists, the ordering of the segments of PERFORMed COBOL paragraphs in the Procedure Division is more critical. This is because the source listing under this condition is a single sequential data set. At present, the suggested sequence is initially by nested level for 2 or 3 levels (depending on the program's complexity) and alphabetically thereafter.

3.1.7.1.2.3 PERFORMed paragraphs should be separated from the main body of code, and from other PERFORMed paragraphs, by at least two blank lines. Logically non-contiguous paragraphs (other than those used in the CASE figure) should be separated by at least one blank line.

3.1.7.1.3 Comments. One of the primary intents of the developers of the COBOL language was to produce a self-documenting language. When this is coupled with the discipline of SP, the resulting programs should be even more readable. Experience has indicated that well written COBOL programs contribute toward meeting this objective. Therefore, it is recommended that the use of comments in the form of NOTE sentences and NOTE paragraphs be held to a minimum. When they are used, they should be organized in such a manner as not to interfere with the readability of the program itself. This may be done by such devices as using blank lines to ensure that the NOTE text stands apart from the program proper and starting and concentrating the textual commentary in the middle of the pages beginning in Column 35-40.

3.1.7.1.4 Indentation and Formatting Conventions. Variables and structures defined in the DATA DIVISION should be arranged in a meaningful order. This order could be alphabetic, by class such as the days of the week, or any other class format. A suggested set of indentation rules for data items is as follows:

3.1.7.1.4.1 General Format. All level 77 and Ø1 variables should have their level numbers in columns 8-9 and names starting in column 12. The PICTURE clause should be between columns 32-45, depending on the length of the longest variable name. All other clauses used should follow the PICTURE clause with normal spacing. If more than one line is needed for a variable's definition, the second and succeeding lines should be indented from the PICTURE clause as follows:

```
77  RECORD-COUNT      PIC 9(7)V99
                        COMP SYNC.
```

3.1.7.1.4.2 Structures. Each successively lower level in a structure should be indented four columns from the next higher level. Level numbers should precede each variable name in the structure on the same line and two columns before it as follows:

```
01  EMPLOYEE-RECORD.
    02  NAME.
        03  FIRST       PIC X(10).
        03  MIDDLE      PIC X.
        03  LAST        PIC X(20).
    02  ADDRESS.
        03  STREET      PIC X(15).
```

When condition-names (level 88 items) are used, they should be indented and written with single spacing between words:

```
        03  TYPE CODE     PIC X.
          88  NEW-TYPE VALUE "B".
          88  OLD-TYPE VALUE "2".
```

3.1.7.1.5 Nested IF. The following should be avoided when using nested IFs:

More than three levels of nesting
Compound conditions
NOTs
Implied subjects and operators

3.1.7.1.6 Module Size. Individual modules should be limited to one printer page or approximately 50 lines of source code whenever possible.

# CHAPTER 4

## FORTRAN PROGRAMING PROCEDURES

4.1 INTRODUCTION. This section used in conjunction with Chapter 1, will be used as a guide for the design and programing of applications software that will be translated by the FORTRAN compiler.

4.2 DESIGN CONSIDERATIONS. Prior to programing of an applicable program, a search should be made for existing programs and subroutines that might be used, either intact or modified. Top Down design should be taken into consideration when designing FORTRAN systems. Although FORTRAN does not lend itself fully to structured programing, the use of structured formatting, one entrance, one exit, top down design with a minimal use of "GO TO" is beneficial to a successful operation.

4.2.1 MODULARITY. Programs should be constructed in modular fashion. They should be broken down into distinct logical segments (subroutines). This practice will make programs far easier to write, test, maintain, and modify. Subroutines and function names when feasible, should be contractions or acronyms that can be related to the processes they are effecting.

4.2.2 LIBRARY FUNCTIONS. Maximum use should be made of basic, existing intrinsic functions and of external functions. The programer can waste valuable time and core storage creating code internal to his routine that could be implemented using functions.

4.2.3 INPUT/OUTPUT FUNCTIONS. I/O functions in FORTRAN programs, as in all other compilers, are best handled by the Executive Control Program (supervisory routine). The applications programer should not attempt to initiate I/O operations directly, but should use the compile provided READ and WRITE statements.

4.2.4 REAL AND INTEGER DATA. The programer should assure that the data is identified with the correct data type. For instance, double precision real data type should not be used if single precision or integer data type would suffice.

4.3 PROGRAM STRUCTURE. Programs consist of statements which are in turn composed of characters grouped according to the following conventions:

4.3.1 SOURCE CARD CODING. FORTRAN statement source cards will use the following card columns. Up to 19 Continuation cards are permitted.

4.3.1 SOURCE CARD CODING. (Cont.)

| COLUMN | USE |
|---|---|
| 1-5 | Statement Label |
| 6 | Continuation if not Ø or blank |
| 7-72 | Statement |
| 73-8Ø | Sequence |

4.3.1.1 Comment Cards. FORTRAN comment source cards use the following card columns:

| COLUMN | USE |
|---|---|
| 1 | C indicates comment |
| 2-72 | Comment verbiage |
| 73-8Ø | Sequence numbers |

4.3.1.2 Sequence numbers. A sequence number, consisting of six digits in the sequence number area, is used to numerically label each card image in a source program to be compiled by the FORTRAN compiler. The use of coded sequence numbers is optional since USACSC Source Library System (SLS) will automatically assign sequence numbers when extracting a source program. However, it is recommended that sequence numbers be assigned when coding. Sequence numbers should be incremented by 1Ø. Alpha characters are strongly recommended for continuation sequence numbering.

4.3.1.3 Statement Labeling. Statements may be labeled (numbered) so that reference may be made to them by other statements. Statement labels are placed in columns 1 through 5 of the statement. Leading zeros are insignificant. From one to five numeric characters may be used. The statement label must be unique within the program unit. Statements should not be labeled unnecessarily. The statement label numbers should increase from physical beginning to physical end of the executable statements. This permits easy following of transfers. Using a separate and distinct block of statement numbers in different sections of the program emphasizes its structure and helps prevent accidental duplication of statement numbers. At the initial writing of a program, leave gaps of about 1ØØ between successively numbered statements to ease later insertions of statements.

4.3.1.4 Statement Ordering. Place specification statements (e.g., DIMENSION, COMMON) at the beginning of the program. This way they are easy to find and do not interrupt the executable statement logic flow. Furthermore, some compilers reject these statements if they are not placed first. Place FORMAT statements where they are easy to find. One method is to group them all at the beginning or end of the program. Also acceptable is to place the FORMAT statement immediately following the I/O statement which referenced it, provided that the FORMAT statement so placed is not multiply addressed.

4.3.1.5 Symbolic Names. Symbolic names are composed of 1-6 alphanumeric characters of which the first must be alphabetic. In naming variables, use names beginning with I through N for integer variables, and names beginning with A through H and O through Z for other variables. This widely accepted convention reduces confusion. A type declaration overrides the implied association. Starting a variable name with characters I through N is an implied (integer, length 4) declaration to the compiler.

4.3.2 FORTRAN CHARACTER SET. The FORTRAN character set consists of the 26 alphabetic characters A through Z, the 10 numeric characters 0 through 9, and the 11 following special characters:

| CHARACTERS | NAME OF CHARACTER |
|---|---|
| | Blank |
| = | Equal |
| + | Plus |
| - | Minus |
| * | Asterisk |
| / | Slash or Virgule |
| ( | Left Parenthesis |
| ) | Right Parenthesis |
| , | Comma |
| . | Decimal Point |
| $ | Currency Sign |

NOTE: In addition to the FORTRAN character set listed in 4.3.2, any other characters acceptable to the applicable processor may be used in HOLLERITH TYPE statements.

4.3.3 OPERATORS USED IN FORTRAN PROGRAMS.

4.3.3.1 Arithmetic Expressions. Arithmetic expressions are formed using the following arithmetic operations:

| OPERATOR | MEANING |
|---|---|
| + | Addition, positive value (zero + element) |
| - | Subtraction, negative value (zero - element) |
| * | Multiplication |
| / | Division |
| ** | Exponentation |

4.3.3.2 Relation Operators. Relation operators are used with two arithmetic expressions and will have a true or false result depending on the relation. The relational operators used are:

| OPERATOR | MEANING |
|---|---|
| .LT. | Less than |
| .LE. | Less than or equal to |
| .EQ. | Equal to |
| .NE. | Not equal to |
| .GT. | Greater than |
| .GE. | Greater than or equal to |

4.3.3.3 Logical Operators. Logical expressions are formed with logical operators and logical elements and have a true or false value.

4.3.3.3 Logical Operators. (Cont.)

The logical operators are:

| OPERATORS | MEANING |
|---|---|
| .OR. | Logical disjunction |
| .AND. | Logical conjunction |
| .NOT. | Logical negation |

4.3.3.4 Additional Information. Use parentheses to make arithmetic expressions completely unambiguous. The expression A**B**C is computed from right to left by some compilers; left to right by others. In general, replace complicated compound expressions by simpler operations when possible. This might be accomplished by single variables previously set to the values of the expressions.

4.4 ARRAYS. Do not needlessly combine into one array what could separate into arrays with fewer dimensions (e.g., use A(1Ø,6), B(1Ø,5) C(1Ø,4) rather than ABC (1Ø,6,3). Similarly, do not needlessly form a single dimension array from what could be single variables. The time and storage required for index manipulation increases as the number of dimensions increase. Arrays shall have the same length in all routines in which they are referenced unless the array is an adjustable array (i.e., the array and its dimensions are dummy arguments). Arrays shall have no more than three dimensions.

4.5 VARIABLE NAMES. Whenever possible use variable names that are relatable to the context of the problem the program is to solve and that correspond to notation or terminology in the block diagram and program documentation. This helps make the listing self-explanatory and relates it to the flow chart and associated documents. Use variable names for quantities that might be expressed as constants but could have to assume another value at a later date.

4.6 PROGRAM COMMENTS.

4.6.1 MEANINGFUL COMMENTS. Make your program self-explanatory by including meaningful comments throughout. Since most programs outlive their authors' responsibility for them and because no computer is permanent, your program will be modified according to new machine, software, or performance requirements. Depending on the complexity of the program, the number of necessary comments varies, but usually the ratio of comments to statements should be at least 1:5. Commentary shall not be included on FORTRAN statements.

4.6.2 IDENTIFICATION OF PROGRAM IN A COMMENT. Identify the program in a comment at the beginning of the listing. Comments should follow this card to provide a program abstract answering such questions as: What does the program do? Is it confined to any particular application? Is it a special version? Why was it written by whom, and when? Is it derived from or directly related to another program? Are any relevant references published?

4.6.3 PROGRAM MODIFICATION. Program modification should be noted by the date and number of the modification (1, 2, 3, ...) and the name of the programer making it, i.e.,

| MODIFICATION NUMBER | DATE OF CHANGE | PROGRAMER NAME |
|---|---|---|
| 3 | YY/MM/DD | H. L. JONES |

4.6.4 PROGRAM COMMENTS FOR SUBROUTINES. For a subroutine, comments describing the calling sequence should follow the identification information. Identify each argument as input, input/output, or output; and explain its purpose, type dimension, etc. The different values that an indicator (such as an error code) can assume should be defined for both input and output.

4.6.5 DISTRIBUTE COMMENT. Distribute comments describing and summarizing the computation appropriately throughout the listing. These should correspond in terminology to the program block diagram. Clever but possibly obscure, coding should be explained in detail. In-line commentary should identify the purpose of every control statement. A control statement is defined as one which conditionally alters a data value or which alters the sequential execution of statements. For ease of reading, comments may be grouped at the beginning of a set of logically contiguous statements. As a minimum, in-line comments should precede blocks of one or more of each of the following:

- IF statements.
- Input/output statements.
- Mixed mode arithmetic assignment statements.
- Call statements
- Control structures.

4.6.6 DESCRIPTIVE COMMENTS. Explain in comments any reason for peculiar array dimensions, e.g., storage limitations or use by other routines.

4.6.7 CONSPICUOUS PRINTING STYLE FOR COMMENT. Use a conspicuous printing style for comments so that they stand out from the rest of the listing. Separate comments from statements by cards that are blank except for the C in Column 1. (Although the listing looks cleaner without the C, some compilers reject totally blank cards.) Comments are further accented if they are indented, starting approximately in Column 15.

4.6.8 RECOVERY PROCEDURES IN COMMENTS. Explain error recovery procedures in comments, unless they are already defined in FORMAT statements. This information is important to those who maintain or modify the program.

4.7 CHECK AND DESK CHECKING.

4.7.1 CHECKOUT METHOD. Plan your checkout method while designing a program. Organize the program so checkout data is easy to prepare. Make up a block diagram and preliminary checkout data before coding. Use the checkout data and block diagram in "desk checking" the program. Caution should be taken that data items that are meant to be real values have not been given a name beginning with I, J, K, L, M, or N. This would cause them to be made integers and is often the cause of execution errors.

4.7.2 DESK CHECKING. Desk checking means manually scrutinizing program logic and deck structure. Mistakes in either can cause an unsuccessful run, so a few minutes of checking is worthwhile.

4.7.3 PROGRAM LOGIC CHECKLIST.

4.7.3.1 Statement Number. Assure that there is a statement number on the statement immediately following each arithmetic IF statement and each GO TO statement.

4.7.3.2 Verify Statement Number. Verify that there are statement numbers for the exits from IF, GO TO, and DO statements.

4.7.3.3 Assure Parentheses Balance. Start from the left with Ø and add 1 for each left parenthesis. If parentheses balance, the count will end up at Ø; however, this does not indicate correct grouping.

4.7.3.4 Subscripted Variables. Every subscripted variable must appear in a specification statement.

4.7.3.5 Check For DO-loop. Check for DO-loop that ends with an IF statement or GO TO statement.

4.7.3.6 Assure Statements Are Present. Assure that all referenced FORMAT statements are present.

4.7.3.7 Check All Hollerith Fields. Check all Hollerith fields for the correct length.

4.7.3.8 CALL Statement. Assure that the number, order, and type of arguments in CALL statements are correct.

4.8 ARGUMENTS.

4.8.1 GROUPING OF ARGUMENTS. For ease of interpretation, group the arguments of a calling sequence in this order: input, input/output, output, error code. An input argument is one whose value the subroutine uses but does not change; an input/output argument is one whose value the subroutine uses and subsequently changes; an output argument is one whose values are computed by the subroutine and where an assignment statement will be found for the output argument name. The error code argument is the means of transmitting diagnostic information to the calling program e.g., whether the subroutine executed normally or abnormally; it is a special kind of output argument.

4.8.2 ERROR CODE. An error code returned by a subroutine should be zero for normal execution and a non-zero value otherwise. The more different non-zero values a subordinate can return, the more specifically it can describe to the calling program the nature of a malfunction or improper condition in the input data.

4.9 NON-INTEGER VARIABLE. The use of non-integer variable or non-integer expressions for array subscripts shall be prohibited.

4.9.1 ARRAY NAMING CONVENTION. In a routine, the appearance of an array name shall be immediately followed by a subscript except in the following cases:

- In the list of an input/output statement.
- In the list of dummy arguments.
- In the list of actual arguments in a reference to an external procedure.
- In a type statement.

4.9.2 ARGUMENTS IN CALL STATEMENTS. Arguments in routine calling statements shall not contain arithmetic or logical expressions.

4.9.3 DATA VARIABLE ASSIGNMENT. Only one data variable assignment shall be made in a line of code (i.e., A=B=C is not allowed).

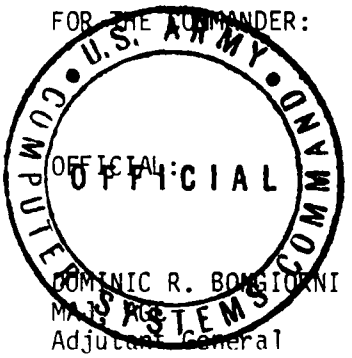
4.9.4 WHOLE NUMBERS. Whole numbers used as exponents shall be expressed as integers.

4.9.5 INPUT/OUTPUT DEVICES. I/O devices shall be referenced by integer variable names.

4.9.6 CONSTANT COUNT INDICES. Constant count indices shall not be used to control the input of data. An integer, real or double precision constant is said to be signed when it is written immediately after a PLUS or MINUS.

The Proponent Agency for this manual is the Technical Evaluation and Standards Directorate. Users are invited to send comments and suggested improvements on DA Form 2028 (Recommended Changes to Publications and Blank Forms) to Commander, USACSC, ATTN: ACSC-TES, Fort Belvoir, VA 22060.

FOR THE COMMANDER:



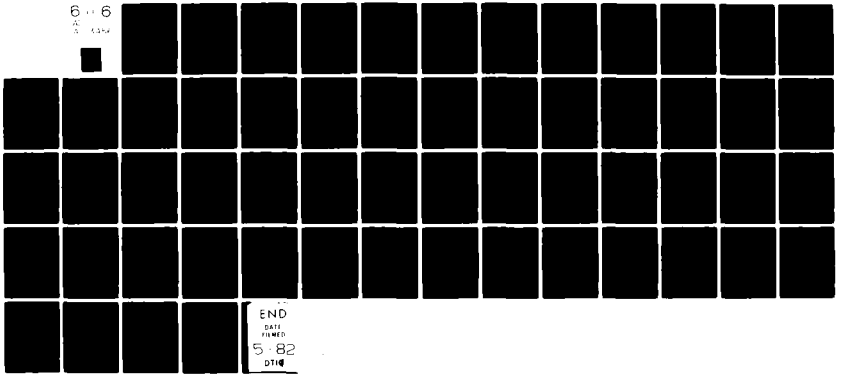

JOHN F. VAN WERT, JR.
COL, GS
Chief of Staff

OFFICIAL:

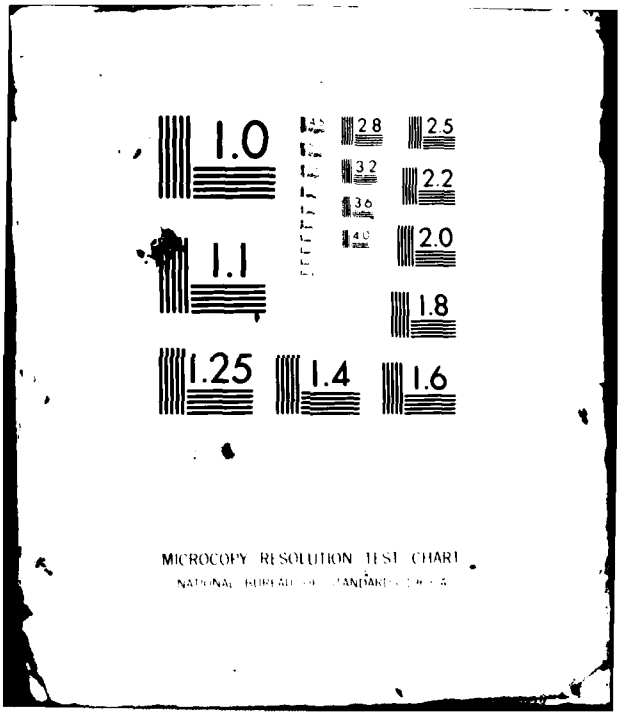
DOMINIC R. BONGIORNI
MAJ, AG
Adjutant General

AD-A113 456 ARMY COMPUTER SYSTEMS COMMAND FORT BELVOIR VA F/G 9/2
PROGRAMING PROCEDURES MANUAL (PPM).(U)
DEC 81

UNCLASSIFIED NL



END
DATE FILMED
5-82
DTIC


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART

# GLOSSARY

The definitions of the COBOL terms in the Glossary are provided merely as reference material or introductory material. The definitions are therefore brief and do not give any detail of syntactical rules.

# GLOSSARY

## DEFINITIONS

ABBREVIATED COMBINED RELATION CONDITION

The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

ACCESS MODE

The manner in which records will be operated upon within a file by the computer.

ACTUAL DECIMAL POINT

The physical representation, using the decimal point characters, period (.) or comma (,), of the decimal point position in a data item. When used, it will appear in a printed report and will require an actual space in storage.

ALPHABET-NAME

A user-defined word, in the SPECIAL-NAMES paragraph of the Environment Division, that assigns a name to a specific character set and/or collating sequence.

ALPHABETIC CHARACTER

A character that belongs to one of the 26 characters of the alphabet or space.

ALPHANUMERIC CHARACTER

Any character in the computer's character set.

ARITHMETIC EXPRESSION

An arithmetic expression can be an identifier or a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression inclosed in parentheses.

ARITHMETIC OPERATORS

The character set that defines COBOL arithmetic operators is as follows:

GLOSSARY (Cont.)

| CHARACTER | MEANING |
|---|---|
| + | ADDITION |
| - | SUBTRACTION |
| * | MULTIPLICATION |
| / | DIVISION |
| ** | EXPONENTIATION |

ASCENDING KEY

A key upon the values of which data is ordered starting with the lowest value of key to the highest value of key, in accordance to the rules of the collating sequence.

ASSUMED DECIMAL POINT

A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

ASYNCHRONOUS PROCESSING

A processing method in which each event or the performance of each operation starts as a result of a signal that the previous operation has been completed.

AT END CONDITION.

A condition caused:

1. During the execution of a READ statement for a sequentially accessed file.

2. During the execution of a RETURN statement, when no next logical record exists for the associated sort file.

BLOCK

A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either continued within the block or that overlap the block. The term is synonymous with physical record.

GLOSSARY (Cont.)

BUFFER

A portion of main storage into which data is read or from which data is written.

BYTE

A generic term to indicate a measurable portion of consecutive binary digits; e.g., an 8-bit or 6-bit byte.

CALLED PROGRAM

A program which is the object of a CALL statement combined at object time with the calling program to produce a run unit.

CALLING PROGRAM

A program which executes a CALL to another program.

CHANNEL

A device that directs the flow of information between the computer main storage and the input/output devices.

CHARACTER

The basic indivisible unit of the language.

CHARACTER POSITION

A character position is the amount of physical storage required to store a single standard data format character whose usage is DISPLAY. Further characteristics of the physical storage are defined by the vendor.

CHARACTER SET

The most basic and indivisible unit of the language is the character. The set of characters used to form a COBOL character-string and separators includes the letters of the alphabet, digits and special characters. The ANSI COBOL character set consists of the characters as defined below.

GLOSSARY (Cont.)

ANSI COBOL

Ø, 1, ....9
A, B, ....Z
+
-
*
/
=
$
;
,
.
"
(
)
>
<
SPACE OR BLANK

In the case of non-numeric literals, comment-entries, and comment lines, the character set is expanded to include the computer's entire character set which may support such non-standard characters as:

!
#
%
&
'
:
?
@

NOTE: USACSC programers should generally use only those characters defined in the ANSI COBOL character set less the < and >. Additional characters may not be printable depending on the printer features and print chains available at a given DPI.

The character set used for the formation of words is restricted as follows:

| | | | |
|---|---|---|---|
| Ø | A | J | S |
| 1 | B | K | T |
| 2 | C | L | U |
| 3 | D | M | V |
| 4 | E | N | W |
| 5 | F | O | X |
| 6 | G | P | Y |
| 7 | H | Q | Z |
| 8 | I | R | - (hyphen) |
| 9 | | | |

NOTE: The hyphen may not be used as the first or last character in a word.

GLOSSARY (Cont.)

CHARACTER-STRING

A sequence of contiguous characters which form a COBOL word, a literal, a PICTURE character-string, or a comment-entry.

CLASS CONDITION

The proposition, for which a truth value can be determined, that the content of an item is entirely alphabetic or entirely numeric.

CLAUSE

A clause is an ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

COLLATING SEQUENCE

The sequence in which the characters that are acceptable to a computer are ordered for purposes of sorting, merging and comparing. The EBCDIC collating sequence, in ascending order, is:

| | | |
|---|---|---|
| 1. | | (space) |
| 2. | . | (Period or decimal point) |
| 3. | < | ("less than" symbol) |
| 4. | ( | (left parenthesis) |
| 5. | + | (plus sign) |
| 6. | $ | (currency symbol) |
| 7. | * | (asterisk) |
| 8. | ) | (right parenthesis) |
| 9. | ; | (semicolon) |
| 10. | - | (hyphen or minus symbol) |
| 11. | / | (stroke, virgule, slash) |
| 12. | , | (comma) |
| 13. | > | ("greater than" symbol) |
| 14. | ' | (apostrophe or single quotation mark) IBM |
| 15. | = | (equal sign) |
| 16. | " | (quotation mark) |
| 17-42. | A through Z | |
| 43-52. | Ø through 9 | |

COLUMN

A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

GLOSSARY (Cont.)

COMBINED CONDITION

A condition that is the result of connecting two or more conditions with the 'AND' or the 'OR' logical operator.

COMMENT-ENTRY

An entry in the Identification Division that may be any combination of characters from the computer character set.

COMMENT LINE

An annotation in the IDENTIFICATION DIVISION or PROCEDURE DIVISION of a COBOL source program. A comment is ignored by the compiler.

COMPILE TIME

The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

COMPILER DIRECTING STATEMENT

A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation.

COMPLEX CONDITION

A condition in which one or more logical operators act upon one or more conditions. (See Negated Simple Condition, Combined Condition, Negated Combined Condition.)

COMPOUND CONDITION

A statement that tests for two or more relational expressions, true or false.

COMPUTER-NAME

A system-name that identifies the computer upon which the program is to be compiled or run.

CONDITION

A status of a program at execution time for which a truth value can be determined. Where the term 'condition' (condition-1, condition-2, ...) appears in these language specifications in or in reference to 'condition' (condition-1, condition-2, ...) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesized, or a combination condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

GLOSSARY (Cont.)

CONDITIONAL EXPRESSION

A simple condition or a complex condition specified in an IF or PERFORM statement. (See Simple Condition and Complex Condition.)

CONDITION-NAME

A user-defined word assigned to a specific value, set of values, or range of values, within the complete set of values that a conditional variable may possess; or the user-defined word assigned to a status of a user-defined switch or device.

CONDITION-NAME CONDITION

The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

CONDITIONAL STATEMENT

A statement which specifies that the determination of a truth value has to be made and any subsequent action of the object program is dependent on this truth value.

CONDITIONAL VARIABLE

A data item that can take on more than one value and the value(s) it assumes is assigned a condition-name.

CONFIGURATION SECTION

A section of the ENVIRONMENT DIVISION that describes overall specifications of source and object computers.

CONNECTIVE

A reserved word that is used to:

1. Associate a data-name, paragraph-name, condition-name or test-name with its qualifier.

2. Link two or more operands written in a series.

3. Form conditions (logical connectives).

CONSOLE

A COBOL mnemonic-name that is used to indicate the console typewriter.

GLOSSARY (Cont.)

CONTIGUOUS ITEMS

Items that are described by consecutive entries in the DATA DIVISION, and that bear a definite hierarchic relationship to each other.

CONTROL BYTES

Bytes used in conjunction with a physical record to serve to identify the record and indicate its length, blocking factor, etc.

CORE STORAGE

CPU storage existing in the form of magnetic cores.

COUNTER

A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

CURRENCY SIGN

The character '$' of the COBOL character set.

CURRENCY SYMBOL

The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present in a COBOL source program, the currency symbol is identical to the currency sign.

CURRENT RECORD

The record which is available in the record area associated with the file.

CURRENT RECORD POINTER

An internal indicator that is used in the selection of the next record.

DATA CLAUSE

A clause that appears in a data description entry in the DATA DIVISION and provides information describing a particular attribute of a data item.

GLOSSARY (Cont.)

DATA DESCRIPTION ENTRY

An entry in the DATA DIVISION that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

DATA DIVISION

This is the third of four divisions of a COBOL program. The files to be used in the program and the records contained within the files are described here.

DATA ITEM

A character or a set of contiguous characters (excluding literals) defined as a unit of data by the COBOL program.

DATA-NAME

A user-defined word that names a data item described in a DATA DESCRIPTION entry in the DATA DIVISION. When used in the General Formats, 'data-name' represents a word which can neither be subscripted, indexed, nor qualified unless specifically permitted by the rules for that format.

DEBUGGING SECTION

A debugging section is a section that contains a USE FOR DEBUGGING statement.

DECLARATIVE-SENTENCE

A compiler-directing sentence consisting of a single USE statement terminated by the separator period.

DECLARATIVES

A set of one or more special-purpose sections, written at the beginning of the PROCEDURE DIVISION, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one or more associated paragraphs.

DELIMITER

A character or a sequence of contiguous characters that identify the end of a string of characters and separates that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

DESCENDING KEY

A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules of the collating sequence.

GLOSSARY (Cont.)

DESTINATION

A symbolic identification of the receiver of a transmission from a queue. A destination is a message used in teleprocessing.

DESTINATION QUEUE

A Message Control Program storage queue for messages to or from remote stations used in teleprocessing.

DEVICE-NUMBER

A number which is assigned to any external device.

DIGIT POSITION

A digit position is the amount of physical storage required to store a single digit. This amount may vary depending on the usage specified in the DATA DESCRIPTION entry that defines the data item.

DIVISION

One or more sections or paragraphs that are formed and combined in accordance with a specific set of rules. Each division consists of the division header and the related division body. There are four divisions in a COBOL program.

IDENTIFICATION

ENVIRONMENT

DATA

PROCEDURE

DIVISION HEADER

A combination of words followed by a period and a space that indicates the beginning of a division. The division headers are:

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

PROCEDURE DIVISION.

GLOSSARY (Cont.)

DYNAMIC ACCESS

An access mode in which specific logical records can be obtained from or placed into a mass storage file in a non-sequential manner and obtained from a file in a sequential manner, during the scope of the same OPEN statement.

EBCDIC CHARACTER

Any one of the symbols included in the eight-bit EBCDIC (Extended Binary-Coded-Decimal Interchange Code) set. All of the 51 characters that belong to the COBOL character set are included.

EDITING CHARACTER

A single character or a fixed two-character combination belonging to the following set:

| CHARACTER | MEANING |
|---|---|
| B | SPACE |
| Ø | ZERO |
| + | PLUS |
| - | MINUS |
| CR | CREDIT |
| DB | DEBIT |
| Z | ZERO SUPPRESS |
| * | CHECK PROTECT |
| $ | CURRENCY SIGN |
| , | COMMA |
| . | PERIOD (DECIMAL POINT) |
| / | STROKE (VIRGULE, SLASH) |

GLOSSARY (Cont.)

ELEMENTARY ITEM

A data item that is described as not being further subdivided logically.

END OF PROCEDURE DIVISION

The physical position of a COBOL source program after which no further procedures appear.

ENTRY

A descriptive set of consecutive clauses terminated by a period and written in the IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, or DATA DIVISION of a COBOL source program.

ENVIRONMENT CLAUSE

A clause that appears as part of an ENVIRONMENT DIVISION entry.

ENVIRONMENT DIVISION

This is the second of the four divisions of a COBOL program. The ENVIRONMENT DIVISION gives information about the computers upon which the source program is compiled and those on which the object program is executed and provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

EXECUTION TIME

The time at which an object program is executed.

EXPONENT

A number that tells the power (number of times it can be factored) of a base number, positive exponents indicate multiplication and negative exponents indicate division. In COBOL, exponentiation is specified by **.

F-MODE RECORDS

Fixed length records.

FIGURATIVE CONSTANT

A compiler-generated value referenced through the use of certain reserved words.

GLOSSARY (Cont.)

FILE

A collection of records.

FILE-CLAUSE

A clause that appears as part of any of the following DATA DIVISION entries.

File Description (FD)

Sort File Description (SD)

Record Description Entry (RD)

FILE-CONTROL

The name of an ENVIRONMENT DIVISION paragraph in which the data files for a given source program are declared.

FILE DESCRIPTION ENTRY

An entry in the FILE SECTION of the DATA DIVISION that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

FILE-NAME

A user-defined word that names a file described in the File Description entry or a Sort File Description entry within the FILE SECTION of the DATA DIVISION.

FILE ORGANIZATION

The permanent logical file structure established at the time that a file is created.

FILE SECTION

The section of the DATA DIVISION that contains File Description entries and Sort File Description entries together with their associated record descriptions.

FORMAT

A specific arrangement of a set of data.

GLOSSARY (Cont.)

FUNCTION-NAME

A name that indicates a specific logical unit, printer and card punch control characters or report codes. In the ENVIRONMENT DIVISION, a function-name can be associated with a mnemonic-name, in order that the mnemonic-name can then be substituted in any valid format.

GROUP ITEM

A named contiguous set of elementary or group items. These items are logically related.

HEADER LABEL

A record that is used to identify the beginning of a physical file or a volume.

HIGH ORDER END

The leftmost character of a string of characters.

I-O-CONTROL

The name of an ENVIRONMENT DIVISION paragraph in which object program requirements for specific input-output techniques, rerun points, sharing of same areas by several data files and multiple file storage on a single input-output device, are specified.

I-O-MODE

The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement for that file.

IDENTIFICATION DIVISION

This is the first of four divisions of a COBOL program. In the IDENTIFICATION DIVISION, you can identify the source program, object program and such documentation as author's name, installation, date-written, etc.

IDENTIFIER

A data-name followed, as required, by the syntactically correct combination of qualifiers, subscripts, and indices necessary to make unique reference to a data item.

GLOSSARY (Cont.)

IMPERATIVE STATEMENT

A statement that begins with an imperative verb and specifies an unconditional action to be taken.

IMPLEMENTOR-NAME

A system-name that refers to a particular feature available on that implementor's computing system.

IN-LINE PROCEDURE

The set of statements that constitutes the main or controlling flow of the run unit.

INDEX

A computer storage area or register, the contents of which represent the identification of a particular element in a table.

INDEX DATA ITEM

A data item in which the values associated with a user defined index-name can be stored in a form specified by the vendor.

INDEX-NAME

A user-defined word that names an index associated with a specific table.

INDEXED DATA-NAME

An identifier that is composed of a data-name, followed by one or more index-names inclosed in parentheses.

INPUT FILE

A file that is opened in the input mode.

INPUT MODE

The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement for that file.

INPUT-OUTPUT FILE

A file that is opened in the I-O mode.

GLOSSARY (Cont.)

INPUT-OUTPUT SECTION

The section of the ENVIRONMENT DIVISION that names the files and the external media required by an object program and which provides information required for transmission and handling of data during execution of the object program.

INPUT PROCEDURE

A set of statements that is executed each time a record is released to the sort file.

INPUT QUEUE

A Message Control Program destination queue from which messages from the remote stations are accepted by the COBOL teleprocessing program.

INPUT-UNIT

A system-name which specifies the input unit from which object program computer instructions are read at object time.

INTEGER

A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point. NOTE: Where the term 'integer' appears in General Formats, 'integer' must not be a numeric data item in the DATA DIVISION, and must not be signed, nor zero unless explicitly allowed by the rules of that format.

KEY

A set of data items which serve to identify the ordering of data.

KEY WORD

A reserved word whose presence is required when the format in which the word appears is used in a source program.

LANGUAGE-NAME

A system-name that specifies a particular programing language.

LEVEL INDICATOR

Two alphabetic characters that identity a specific type of file or a position in a hierarchy. The level indicators are: FD, SD, RD, CD.

GLOSSARY (Cont.)

LEVEL-NUMBER

A user-defined word which indicates the position of a data item in the hierarchical structure of a logical record or which indicates special properties of a DATA DESCRIPTION entry. A level-number is expressed as a one or two digit number. Level-numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-number 77 identifies special properties of a DATA DESCRIPTION entry.

LIBRARY-NAME

A user-defined word that names a COBOL library that is to be used by the compiler for a given source program compilation.

LIBRARY TEXT

A sequence of character-strings and/or separators in a COBOL library.

LINE NUMBER

An integer that denotes the vertical position of a report line on a page.

LITERAL

A character-string whose value is implied by the ordered set of characters comprising the string.

LOGICAL OPERATOR

One of the reserved words AND, OR, or NOT. AND and OR may be used as logical connectives. NOT can be used for logical negation.

LOGICAL RECORD

The most inclusive data item. The level-number for a record is Ø1.

LOW-ORDER END

The rightmost character of a string of characters.

GLOSSARY (Cont.)

MASS STORAGE

A storage medium in which data may be organized and maintained in both a sequential and nonsequential manner.

MASS STORAGE CONTROL SYSTEM (MSCS)

An input-output control system that directs, or controls, the processing of mass storage files.

MASS STORAGE FILE

A collection of records that is assigned to a mass storage medium.

MNEMONIC-NAME

A user-defined word that is associated in the ENVIRONMENT DIVISION with a specific implementor-name.

MODE-NAME

A system-name that refers to a particular method of data representation on a physical storage medium.

NATIVE CHARACTER SET

The implementor-defined character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

NATIVE COLLATING SEQUENCE

The implementor-defined collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

NEGATED COMBINED CONDITION

The "NOT" logical operator immediately followed by a parenthesized combined condition.

NEGATED SIMPLE CONDITION

The "NOT" logical operator immediately followed by a simple condition.

NEXT EXECUTABLE SENTENCE

The next sentence to which control will be transferred after execution of the current statement is complete.

GLOSSARY (Cont.)

NEXT RECORD

The record which logically follows the current record of a file.

NONCONTIGUOUS ITEMS

Elementary data items, in the WORKING-STORAGE which bear no hierarchic relationship to other data items.

NON-NUMERIC ITEM

A data item whose description permits its contents to be composed of any combination of characters taken from the computer's character set. Certain categories of non-numeric items may be formed from more restricted character sets.

NON-NUMERIC LITERAL

A literal bounded by quotation marks. The string of characters may include any character in the computer's character set. To represent a single quotation mark character within a non-numeric literal, two contiguous quotation marks must be used.

NONSWITCHED LINE

A line that is a continuous link between a remote station and the computer, in teleprocessing.

NUMERIC CHARACTER

A character that belongs to the set of digits: Ø through 9.

NUMERIC EDITED CHARACTER

A numeric character which may be used in a printed output. It may consist of external decimal digits 'Ø' through '9', the decimal point, commas, the dollar sign, etc.

GLOSSARY (Cont.)

NUMERIC ITEM

A data item whose description restricts its contents to a value represented by characters chosen from the digits 'Ø' through '9'. If signed, the item may also contain a '+', '-', or other representation of an operational sign.

NUMERIC LITERAL

A literal composed of one or more numeric characters that may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

OBJECT-COMPUTER

The name of an ENVIRONMENT DIVISION paragraph in which the computer upon which the object program will be run is described.

OBJECT OF ENTRY

A set of operands and reserved words, within a DATA DIVISION entry, that is coded immediately following the subject of the entry.

OBJECT PROGRAM

A set or group of executable machine language instructions and other material designed to interact with data to provide problem solutions. An object program is generally the machine language result of the operation of a COBOL compiler on a source program.

OBJECT TIME

The time at which an object program is executed.

OPEN MODE

The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, or I-O.

GLOSSARY (Cont.)

OPERAND

Any lower-case word (or words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

OPERATIONAL SIGN

An algebraic sign, associated with a numeric data item or a numeric literal to indicate whether its value is positive or negative.

OPTIONAL WORD

A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

OUT-OF-LINE PROCEDURE

A set of statements not included in the main or controlling flow of the program.

OUTPUT FILE

A file that is opened in either the output mode or extend mode.

OUTPUT MODE

The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified, for that file and before the execution of a CLOSE statement for that file.

OUTPUT PROCEDURE

A set of statements to which control is given during execution of a SORT statement after the sort function is completed.

OUTPUT QUEUE

An MCP (Message Control Program) destination queue into which a COBOL teleprocessing program places messages for one or more remote stations.

OVERFLOW CONDITION

A condition which occurs in string manipulation when the sending area(s) contain untransferred characters after the receiving area(s) have been filled.

GLOSSARY (Cont.)

OVERLAY

The use of the same areas of internal storage for different stages in processing a problem.

PAGE

A vertical division of a report representing a physical separation of report data, the separation being based on internal reporting requirements and/or external characteristics of the reporting medium.

PARAGRAPH

In the PROCEDURE DIVISION, a paragraph-name followed by a period and a space and by zero, one or more sentences. In the IDENTIFICATION and ENVIRONMENT DIVISIONS, a paragraph header followed by one or more entries.

PARAGRAPH HEADER

A reserved word, followed by a period and a space that indicates the beginning of a paragraph in the IDENTIFICATION and ENVIRONMENT DIVISIONS. The permissible paragraph headers are:

| In the IDENTIFICATION DIVISION | In the ENVIRONMENT DIVISION |
|---|---|
| PROGRAM-ID. | SOURCE-COMPUTER. |
| AUTHOR. | OBJECT-COMPUTER. |
| INSTALLATION. | SPECIAL-NAMES. |
| DATE-WRITTEN. | FILE-CONTROL. |
| DATE-COMPILED. | I-O-CONTROL. |
| SECURITY. | |

PARAGRAPH-NAME

A user-defined word that identifies and begins a paragraph in the PROCEDURE DIVISION.

PARAMETER

A variable that is given a specific value and is used to pass data values between calling and called programs.

GLOSSARY (Cont.)

PHRASE

A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or of a COBOL clause.

PHYSICAL RECORD

A term used synonymously with the term BLOCK. A physical record can be composed of a portion of one logical record, of one complete logical record, or of a group of logical records.

PRIORITY-NUMBER

A number which classifies source program sections in the PROCEDURE DIVISION. A priority-number ranges in values from Ø to 99.

PROCEDURE

A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the PROCEDURE DIVISION.

PROCEDURE DIVISION

One of the four main parts of a COBOL program. The PROCEDURE DIVISION contains instructions for solving a problem.

PROCEDURE-NAME

A user-defined word which is used to name a paragraph or section in the PROCEDURE DIVISION. It consists of a paragraph-name (which may be qualified), or a section-name.

PROCESSING CYCLE

A single execution of a defined out-of-line procedure.

PROGRAM-NAME

A user-defined word that identifies a COBOL source program.

PSEUDO-FILE-NAME

A user-defined word that names a file residing on a multiple file tape for which no FILE DESCRIPTION entry is specified.

GLOSSARY (Cont.)

PSEUDO-TEXT

A sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters.

PSEUDO-TEXT DELIMITER

Two contiguous equal sign (=) characters used to delimit pseudo-text.

PUNCTUATION CHARACTERS

The character set consists of the punctuation characters as follows:

| GRAPHIC | NAME |
|---|---|
| , | COMMA |
| ; | SEMICOLON |
| . | PERIOD |
| " | QUOTATION MARK |
| ( | LEFT PARENTHESIS |
| ) | RIGHT PARENTHESIS |
| | SPACE |
| = | EQUAL |

QUALIFIED DATA-NAME

An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

QUALIFIER

A data-name which is used in a reference together with another data-name at a lower level in the same hierarchy.

A section-name which is used in a reference together with a paragraph-name specified in that section.

A library-name which is used in a reference together with a text-name associated with that library.

QUEUE

A logical collection of messages awaiting transmission or processing.

GLOSSARY (Cont.)

QUEUE BLOCKS

Blocks containing status and control information pertaining to the message being processed and to each active queue. Queue blocks are created when a queue is first accessed by a COBOL teleprocessing run unit and are chained together when in one region/partition.

QUEUE NAME

A symbolic name that indicates to the MCS (Message Control System) the logical path by which a message or a portion of a completed message may be accessible in a queue.

RECORD

A set of data items grouped for handling internally or by input/output systems.

REEL

A module of external storage which is associated with a tape device.

REFERENCE FORMAT

A format that provides a standard method for describing COBOL source programs.

RELATION CHARACTER

A character that belongs to the following set:

| CHARACTER | MEANING |
|---|---|
| > | GREATER THAN |
| < | LESS THAN (NOTE: > and < will not be used in USACSC COBOL.) |
| = | EQUAL TO (NOTE: To be used only in the COMPUTE statement.) |

RELATION CONDITION

The proposition, for which a truth value can be determined, that the value of an arithmetic expression or data item has a specific relationship to the value of another arithmetic expression or data item.

GLOSSARY (Cont.)

RELATIONAL OPERATOR

A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and the relation characters used in the construction of a relation condition. The permissible operators and their meanings are:

| RELATIONAL OPERATOR | MEANING |
|---|---|
| IS (NOT) GREATER THAN | GREATER THAN OR NOT GREATER THAN |
| IS (NOT) LESS THAN | LESS THAN OR NOT LESS THAN |
| IS (NOT) EQUAL TO | EQUAL TO OR NOT EQUAL TO |
| EQUAL | EQUAL TO |

REMOTE STATION

A control unit and one or more input/output devices used in teleprocessing which are connected to the central computer through common carrier facilities. A remote station may be either a terminal device or another computer.

REPORT

A presentation of a set of processed data.

RESERVED WORD

A COBOL word specified in the list of words which may be used in a COBOL source program, but which must not appear in the programs as user-defined words or system-names.

ROUTINE-NAME

A user-defined word that identifies a procedure written in a language other than COBOL.

RUN UNIT

A set of one or more object programs which function, at object time, as a unit to provide problem solutions.

GLOSSARY (Cont.)

SECTION

A set of zero, one, or more paragraphs or entries, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

SECTION HEADER

A combination of words followed by a period and a space that indicates the beginning of a section in the ENVIRONMENT, DATA, or PROCEDURE DIVISION.

In the ENVIRONMENT and DATA DIVISION, a section header is composed of reserved words followed by a period and a space. The permissible section headers are:

| In the ENVIRONMENT DIVISION | In the DATA DIVISION |
|---|---|
| CONFIGURATION SECTION. | FILE SECTION. |
| INPUT-OUTPUT SECTION. | WORKING-STORAGE SECTION. |

In the PROCEDURE DIVISION, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a segment-number (optional), followed by a period and a space.

SECTION-NAME

A user-defined word which names a section in the PROCEDURE DIVISION.

SEGMENT-NUMBER

A user-defined word which classifies sections in the PROCEDURE DIVISION for purposes of segmentation. Segment-numbers may contain only the characters '0', '1', ...'9'. A segment-number may be expressed either as a one or two digit number.

SENTENCE

A sequence of one or more statements, the last of which is terminated by a period followed by a space.

SEPARATOR

A punctuation character used to delimit a character-string.

GLOSSARY (Cont.)

SEQUENTIAL ACCESS

An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

SEQUENTIAL FILE

A file with sequential organization.

SEQUENTIAL ORGANIZATION

The permanent logical file structure in which a record is identified by a predecessor-to-successor relationship established when the record is placed into the file.

SEQUENTIAL PROCESSING

A term used synonymously with SYNCHRONOUS PROCESSING.

SIGN CONDITION

The proposition for which a truth value can be determined that the algebraic value of a data item or an arithmetic expression is either positive, negative, or equal to zero.

SIMPLE CONDITION

Any single condition chosen from the set:

relation condition

class condition

condition-name condition

switch-status condition

sign condition

message condition

SLACK BYTES

Bytes which are inserted between data items or between records in order to properly align some numeric items. Slack bytes are in some cases automatically inserted by the compiler and in some cases is the responsibility of the programer to insert them. The SYNCHRONIZED clause instructs the compiler to insert slack bytes for proper alignment.

GLOSSARY (Cont.)

SORT FILE

A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

SORT-FILE NAME

A data-name which is used to identify a sort file.

SORT-KEY

The fields in a record which determine, or are used as a basis for determining, the sequence of records in a file.

SORT-WORK-FILE

A collection of records used in a sorting operation as it exists on an intermediate device(s).

SORT MERGE FILE DESCRIPTION ENTRY

An entry in the FILE SECTION of the DATA DIVISION that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

SOURCE

The symbolic identification of the originator of a transmission to a queue.

SOURCE-COMPUTER

The name of an ENVIRONMENT DIVISION paragraph in which the computer environment, within which the source program is compiled, is described.

SOURCE ITEM

An identifier designated by a SOURCE clause that provides the value of a printable item.

SOURCE PROGRAM

A syntactically correct set of COBOL statements beginning with an IDENTIFICATION DIVISION and ending with the end of the PROCEDURE DIVISION.

GLOSSARY (Cont.)

SPECIAL CHARACTER

A character that belongs to the following set:

| CHARACTER | MEANING |
|---|---|
| + | PLUS SIGN |
| - | MINUS SIGN |
| * | ASTERISK |
| / | SLASH |
| = | EQUAL SIGN |
| $ | CURRENCY SIGN |
| , | COMMA |
| ; | SEMICOLON |
| . | PERIOD |
| " | QUOTATION MARK |
| ( | LEFT PARENTHESIS |
| ) | RIGHT PARENTHESIS |
| > | GREATER THAN SYMBOL (Not USACSC COBOL) |
| < | LESS THAN SYMBOL (Not USACSC COBOL) |
| ! | EXCLAMATION POINT |
| # | NUMBER SIGN |
| % | PERCENT |
| & | AMPERSAND |
| ' | APOSTROPHE |
| : | COLON |
| ? | QUESTION MARK |
| @ | COMMERCIAL AT |

SPECIAL-CHARACTER WORD

A reserved word which is an arithmetic operator or a relation character.

SPECIAL-NAMES

The name of an ENVIRONMENT DIVISION paragraph in which implementor-names are related to user-specified mnemonic-names.

SPECIAL REGISTERS

Certain compiler generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL features.

STANDARD DATA FORMAT

The concept used in describing data in a COBOL DATA DIVISION under which the characteristics or properties of the data are expressed in a form oriented

GLOSSARY (Cont.)

to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

STATEMENT

A syntactically valid combination of words and symbols written in the PROCEDURE DIVISION beginning with a verb.

SUB-QUEUE

A logical hierarchical division of a queue.

SUBJECT OF ENTRY

An operand or reserved word that appears immediately following the level indicator or the level-number in the DATA DIVISION entry.

SUBPROGRAM

A term used synonymously with the term CALLed program.

SUBSCRIPT

An integer whose value identifies a particular element in a table.

SUBSCRIPTED DATA-NAME

An identifier that is composed of a data-name followed by one or more subscripts inclosed in parentheses.

SWITCH-STATUS CONDITION

The proposition, for which a truth value can be determined, that an implementor-defined switch, capable of being set to an 'on' or 'off' status, has been set to a specific status.

SWITCHED LINE

A communication line, used in teleprocessing, for which no single continuous path between the central computer and the remote station exists. Several alternative paths are also available for transmission. The common carrier switching equipment is used to select the proper path.

GLOSSARY (Cont.)

SYNCHRONOUS PROCESSING

A processing method in which each event or the performance of each operation starts as a result of a signal generated by a clock (contrasted with asynchronous processing).

SYSTEM-NAME

A COBOL word which is used to communicate with the operating environment.

TABLE

A set of logically consecutive items of data that are defined in the DATA DIVISION by means of the OCCURS clause.

TABLE ELEMENT

A data-item that belongs to the set of repeated items comprising a table.

TERMINAL

The originator of a transmission to a queue, or the receiver of a transmission from a queue.

TEST CONDITION

A statement which may be either true or false depending on the existing circumstance at the time of the test.

TEXT-NAME

A user-defined word which identifies library text.

TEXT-WORD

Any character-string or separator except space, in a COBOL library or in pseudo-text.

TRAILER LABEL

A record used to indicate the ending of a physical file or of a volume.

TRUTH VALUE

The representation of the result of the evaluation of a condition in terms of one of two values: True or false.

GLOSSARY (Cont.)

U-MODE RECORDS

Records of which the length is unspecified. They may be either fixed or variable. There is only one record per block.

UNARY OPERATOR

A plus (+) or a minus (-) sign, which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression by +1 or -1 respectively.

UNIT

A module of mass storage, the dimensions of which are determined by each implementor.

USER-DEFINED WORD

A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

V-MODE RECORDS

Records of which the length is variable. Blocks may contain more than one record. Each record contains a field which specifies its length and each block contains a field to specify its length.

VARIABLE

A data-item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

VARIABLE-LENGTH DATA ITEM

A variable-length data item is a data item which, although physically fixed in size, contains a logically variable number of characters.

VARIABLE-OCCURRENCE DATA ITEM

A variable occurrence data item is a table element which is repeated a variable number of times. Such an item must contain an OCCURS clause or be subordinate to such an item.

GLOSSARY (Cont.)

VERB

A word that expresses an action to be taken by a COBOL compiler or object program.

VOLATILITY

In the case of a volatile file, the quality of being changeable, transient, temporary or unpredictable in nature.

VOLUME

A module which is externally stored. For tape, it is a reel; for mass storage devices, it is a unit.

VOLUME SWITCH PROCEDURES

Automatic procedures that are executed at the end of a reel or unit before an end-of-file condition is reached.

WORD

A character-string of not more than 30 characters which forms a user-defined word, a system-name, or a reserved word.

WORKING-STORAGE SECTION

The section of the DATA DIVISION that describes WORKING-STORAGE data items and constants, composed either of noncontiguous items or of WORKING-STORAGE records or of both.

77-Level-Description-Entry

A data description entry that describes a noncontiguous data item with the level-number 77.

## RESERVED WORDS

This list of words includes the reserved words published in ANSI X3.23-1974, and the reserved words designated by IBM, CDC, and Honeywell in their various COBOL compilers. These words should not be used except as provided for in this manual to maintain program transferability between vendors' ADP hardware and compilers.

RESERVED WORDS (Cont.)

ABOUT
ACCEPT
ACCEPT-CARD-READER
ACCEPT-CONSOLE
ACCESS
ACCESS-PRIVACY
ACCOUNTING
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALPHANUMERIC
ALPHANUMERIC-EDITED
ALSO
ALTER
ALTERNATE
AN
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
AUX-CHANNEL
AUX-CHANNELS
BASIS
BCD
BEFORE
BEGINNING
BEGIN/PROG/AT
BINARY
BIT
BITS
BLANK
BLOCK
BLOCKS
BOTTOM
BUILD
BY
CALL
CALLEE-TASK
CANCEL
CARD-PUNCH
CD
CDC
CF
CH
CHANGED
CHANNEL
CHANNELS
CHARACTER
CHARACTERS
CHECK
CLASS
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COM-REG
COMMA
COMMON-STORAGE
COMMON-W-STORAGE
COMMUNICATION
COMP
COMP-1
COMP-2
COMP-3
COMP-4
COMP-n
COMPASS
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTATIONAL-4
COMPUTE
COMPUTER
CONFIGURATION
CONSOLE
CONSOLE-KEYBOARD
CONSOLE-TYPEWRITER

RESERVED WORDS (Cont.)

CONSTANT
CONTAINS
CONTIGUOUS
CONTINUE/PROG/AT
CONTROL
CONTROLS
COPY
CORE-INDEX
CORR
CORRESPONDING
COUNT
CRT
CSP
CURRENCY
CURRENT-DATE
CYL-INDEX
CYL-OVERFLOW
CØ1
CØ2
CØ3
CØ4
CØ5
CØ6
CØ7
CØ8
CØ9
C1Ø
C11
C12
DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK
DE
DEBUG
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUG-SWITCH
DEBUGGING
DECIMAL
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DENSITY
DEPENDING
DEPTH
DESCENDING
DESTINATION
DETAIL
DIGIT
DIGITS
DISABLE
DISK
DISP
DISPLAY
DISPLAY-CONSOLE
DISPLAY-*n*
DISPLAY-PRINTER
DISPLAY-ST
DIVIDE
DIVISION
DOLLAR
DOUBLE-BUFFER
DOUBLE-FLOAT
DOWN
DUPLICATES
DYNAMIC
EDITION NUMBER
EGI
EJECT
ELSE
EMI
ENABLE
END
END-OF-PAGE
ENDING
ENTER

RESERVED WORDS (Cont.)

ENTRY
ENVIRONMENT
EOP
EQUAL
EQUALS
ERROR
ESI
ETI
EVERY
EVF-SIGNAL
EXAMINE
EXCEEDS
EXCEPTION
EXHIBIT
EXIT
EXOR
EXTEND
EXTENDED-SEARCH
FD
FILE
FILE-CONTROL
FILE-ID
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIND
FINIS
FIRST
FLOAT
FLOATING-POINT
FOOTING
FOR
FROM
GEN-ND
GENERATE
GENERATION-NUMBER
GET
GIVING
GO
GOBACK
GREATER
GROUP
H-nnn
H-2ØØ-SPECIAL
H-2ØØØ-SPECIAL
HEADING
HIGH
HIGH-VALUE
HIGH-VALUES
HLT-CTL
HOLD
HONEYWELL-nnn
HVF-SIGNAL
HYPER
I-O
I-O-CONTROL
ID
IDENTIFICATION
IF
IN
INCLUDE
INDEX
INDEX-n
INDEXED
INDEXED-SEQUENTIAL
INDICATE
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
INPUT-RECOVERY
INSERT
INSPECT
INSTALLATION
INTO
INVALID
IS
JUST
JUSTIFIED
KEY
KEYS
LABEL
LABEL-RETURN
LAST
LEADING

RESERVED WORDS (Cont.)

LEAVE
LEAVING
LEFT
LENGTH
LESS
LIBRARY
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LINKED-INDEX
LOAD
LOADER
LOCATION
LOCK
LOW
LOW-VALUE
LOW-VALUES
LOWER-BOUND
LOWER-BOUNDS
MASS-STORAGE-DEVICE
MASS-STORAGE-DEVICES
MASTER-INDEX
MEMORY
MERGE
MESSAGE
MODE
MODIFICATION-PRIVACY
MODULE
MODULES
MORE-LABELS
MOVE
MSD
MULTIPLE
MULTIPLY
MULTIPLY-DIVIDE
NAMED
NATIVE
NEAC-280
NEGATIVE
NEXT
N-nnn
NO
NO-TAPE-MARK
NOMINAL
NOT
NOTE
NSTD-REELS
NUMBER
NUMERIC
NUMERIC-EDITED
OBJECT-COMPUTER
OBJECT-PROGRAM
OCCURS
OF
OFF
OH
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OTHERWISE
OUTPUT
OUTPUT-RECOVERY
OV
OVERFLOW
OWNER-ID
PAGE
PAGE-COUNTER
PAPER-TAPE-PUNCH
PAPER-TAPE-READER
PARTITIONED
PASSWORD
PERFORM
PERMIT
PF
PH
PIC
PICTURE
PLACE

RESERVED WORDS (Cont.)

PLACES
PLUS
POINT
POINTER
POSITION
POSITIONING
POSITIVE
PREPARED
PRINT-SWITCH
PRINTER
PRINTERS
PRINTING
PRIORITY
PROCEDURE
PROCEDURES
PROCEED
PROCESS
PROCESSING
PROGRAM
PROGRAM-ID
PROTECT
PT-PUNCH
PT-READER
PUNCH
PUNCHES
QUEUE
QUOTE
QUOTES
R-C INDEX
RANDOM
RNAGE
RD
READ
READER
READER-PUNCH
READER-PUNCHES
READY
REASSIGNMENT
RECEIVE
RECORD
RECORD-MARK
RECORD-OVERFLOW
RECORDING
RECORDS
REDEFINES
REEL
REEL-NUMBER
REFERENCES
REL-CODE
RELATIVE
RELEASE
RELOAD
RELOCATABLE-CODE
REMAINDER
REMARKS
REMOVAL
RENAMES
RENAMING
REORG-CRITERIA
REPLACE
REPLACING
REPORT
REPORTING
REPORTS
REPREAD
RERUN
RESERVE
RESET
RESIDENT-CYLINDER-INDEX
RETENTION-CYCLE
RETURN
RETURN-CODE
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN
RWCS
SA
SAME
SCRATCH
SD
SEARCH

RESERVED WORDS (Cont.)

SECTION
SECTOR
SECURITY
SEEK
SEGMENT
SEGMENTATION
SEGMENTED
SELECT
SELECTED
SEND
SENSE-SWITCH
SENTENCE
SEPARATE
SEQUENCE
SEQUENCED
SEQUENTIAL
SERVICE
SET
SET-ID
SHORT-GAP
SIGN
SIGNED
SIZE
SKIP1
SKIP2
SKIP3
SORT
SORT-CORE-SIZE
SORT-FILE-SIZE
SORT-MERGE
SORT-MESSAGE
SORT-MODE-SIZE
SORT-RETURN
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
STANDARD-8Ø
STANDARD-12Ø
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUB-QUEUE-n
SUBTRACT
SUM
SWITCH
SUPERVISOR
SUPPRESS
SUSPEND
SYMBOLIC
SYNC
SYNCHRONIZED
SYSIN
SYSIPT
SYSLST
SYSOUT
SYSPCH
SYSPUNCH
SYSTEM-DATE
SYSTEM-INPUT
SYSTEM-OUTPUT
SYSTEM-PUNCH
SYSTEM-TIME
SØ1
SØ2
TABLE
TALLY
TALLYING
TAPE
TAPE-UNIT
TAPE-UNITS
TERMINAL
TERMINATE
TEXT
THAN
THEN
THAN
THEN
THROUGH
THRU
TIME
TIME-OF-DAY

RESERVED WORDS (Cont.)

TIMES
TO
TOP
TOTALED
TOTALING
TRACE
TRACK
TRACK-AREA
TRACK-LIMIT
TRACKS
TRAILING
TRAILING-COUNT
TRANSFORM
TTY
TYPE
UNBANNERED
UNEQUAL
UNIT
UNIT-OF-ALLOCATION
UNITS-OF-ALLOCATION
UNSTRING
UNTIL
UP
UPON
UPPER-BOUND
UPPER-BOUNDS
USAGE
USE
USING
VALUE
VALUES
VARYING
VOL-S-NO
VOLUME-SERIAL-NUMBER
VERIFICATIONL-NUMBER
VOLUME
VOLUMES
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
WRITE-CHECKED
WRITE-ONLY
WRITE-VERIFY
ZERO
ZEROES
ZEROS
*
**
+
-
/
=

# INDEX

This index provides an alphabetical list of paragraph headings or titles entered in the text of this manual. It is provided as an easy reference and a primary source of information to enable and assist the USACSC programer to readily locate specific data in the text.

# INDEX


| | Paragraph | Page |
|---|---|---|
| ACCEPT Statement | 2.4.9.1 | 2-121 |
| ACCESS Clause | 2.4.6.11 | 2-47 |
| ADD Statement | 2.4.9.2 | 2-123 |
| Additional Information | 4.3.3.4 | 4-5 |
| Additional Recommended Coding Conventions | 3.1.7.1 | 3-18 |
| Address Pointer | 2.9.2.2 | 2-420 |
| Algebraic Signs | 2.3.4.5 | 2-8 |
| ALTER Statement | 2.4.9.3 | 2-126 |
| ALTERNATE RECORD KEY Clause | 2.4.6.14 | 2-51 |
| ANSI COBOL | 2.1 | 2-1 |
| ANSI COBOL, Authority to Grant an Exception | 1.4.4 | 1-3 |
| ANSI COBOL, Exception to the Use of | 1.4.3 | 1-3 |
| APPLY Clause | 2.9.8 | 2-422 |
| Arguments | 4.8 | 4-7 |
| Arguments in CALL Statements | 4.9.2 | 4-8 |
| Arithmetic Expressions | 2.4.8.3.3 | 2-103 |
| Arithmetic Expressions | 4.3.3.1 | 4-4 |
| Arithmetic Operations | 2.7.3.4 | 2-402 |
| Arithmetic Operators | 2.4.8.3.4 | 2-106 |
| Array Naming Convention | 4.9.1 | 4-8 |
| Arrays | 4.4 | 4-5 |
| ASSIGN Clause | 2.4.6.9 | 2-35 |
| Assure Parentheses Balance | 4.7.3.3 | 4-7 |
| Assure Statements Are Present | 4.7.3.6 | 4-7 |
| AUTHOR Paragraph | 2.4.5.3 | 2-24 |
| Authority to Grant an Exception | 1.4.4 | 1-3 |
| Authority to Grant an Exception | 1.5.3 | 1-4 |
| Backup Programer | 3.1.3.1 | 3-1 |
| Blank Lines | 2.3.5.5 | 2-14 |
| BLANK WHEN ZERO Clause | 2.4.7.23 | 2-98 |
| BLOCK CONTAINS Clause | 2.4.7.9 | 2-71 |
| Braces, Element | 2.4.3.2.5 | 2-18 |
| Brackets, Function | 2.4.3.2.6 | 2-18 |
| Branching Statements | 2.7.3.5 | 2-405 |
| CALL Statement | 2.4.9.4 | 2-127 |
| CALL Statement | 4.7.3.8 | 4-7 |
| CANCEL Statement | 2.4.9.5 | 2-129 |
| Cancel, System Action Under | 2.5.6.6 | 2-285 |
| Card | 1.11.6.3 | 1-10 |
| CASE Statement | 2.4.9.6 | 2-130 |
| CASE Statement | 2.5.1.5 | 2-233 |
| Catalogued Programs | 2.7.6.2 | 2-418 |

| | Paragraph | Page |
|---|---|---|
| Changes to Manual | 1.6 | 1-4 |
| Chapter 1 | 1.2.1 | 1-1 |
| Chapter 2 | 1.2.2 | 1-1 |
| Chapter 3 | 1.2.3 | 1-2 |
| Chapter 4 | 1.2.4 | 1-2 |
| Character-String | 2.3.2.4 | 2-2 |
| Check All Hollerith Fields | 4.7.3.7 | 4-7 |
| Check And Desk Checking | 4.7 | 4-7 |
| Check For DO-loop | 4.7.3.5 | 4-7 |
| Checkout Method | 4.7.1 | 4-7 |
| Chief Programer | 3.1.3.2 | 3-2 |
| Chief Programer Team (CPT) | 3.1.4.6 | 3-14 |
| Chief Programer Team, Team Operation or | 3.1.3.15 | 3-4 |
| Classes of Data, Concept of | 2.3.4.4 | 2-7 |
| Clause | 2.3.5.3.6 | 2-13 |
| Clause | 2.3.6.1.7 | 2-15 |
| Clauses | 2.3.5.3.8 | 2-13 |
| CLOSE Statement | 2.4.9.7 | 2-131 |
| COBOL Levels, Concept of | 2.3.4.2 | 2-6 |
| COBOL Program Debugging Aids, DOS | 2.5.6.2 | 2-271 |
| COBOL Program Debugging Aids, OS | 2.5.6.11 | 2-326 |
| COBOL Program Structure | 2.4.4 | 2-20 |
| COBOL Program Structure Techniques | 2.7 | 2-381 |
| COBOL Program, Structure of the | 2.4.4.3 | 2-21 |
| COBOL Segmentation Facility | 2.5.2 | 2-235 |
| COBOL Specifications, USACSC | 2.4 | 2-16 |
| Code Walkthrough | 3.1.4.5.7.3 | 3-14 |
| CODE-SET Clause | 2.4.7.10 | 2-73 |
| Coding | 2.3.6.1 | 2-14 |
| Coding Conventions, USACSC Standard | 2.3.6 | 2-14 |
| Coding Data Description Entries | 2.3.6.1.6 | 2-15 |
| Coding Paragraph/Section Names | 2.3.6.1.5 | 2-15 |
| Coding, Single Source Library System | 2.8.3 | 2-419 |
| Comment Cards | 4.3.1.1 | 4-2 |
| Comment Lines | 2.3.5.6 | 2-14 |
| Comments | 3.1.7.1.3 | 3-19 |
| Common Causes of Errors | 2.5.6.3 | 2-272 |
| Commonly Encountered User Errors | 2.5.6.8 | 2-297 |
| Compatibility, OS/DOS | 2.9 | 2-420 |
| Completion Code - ØØ1 | 2.5.6.12 | 2-326 |
| Completion Code - Ø13 | 2.5.6.13 | 2-328 |
| Completion Code - Ø31 | 2.5.6.14 | 2-328 |
| Completion Code - Ø3B | 2.5.6.15 | 2-330 |
| Completion Code - Ø3D | 2.5.6.16 | 2-331 |
| Completion Code, OCx Note | 2.5.6.17 | 2-331 |

| | Paragraph | Page |
|---|---|---|
| Completion Code - OC1 | 2.5.6.18 | 2-331 |
| Completion Code - OC5 | 2.5.6.19 | 2-332 |
| Completion Code - OC7 | 2.5.6.20 | 2-333 |
| Completion Code - 237 | 2.5.6.21 | 2-334 |
| Completion Code - 637 | 2.5.6.22 | 2-336 |
| Completion Code - 8Ø4 | 2.5.6.23 | 2-338 |
| Completion Code - 8Ø6 | 2.5.6.24 | 2-338 |
| Completion Code - 813 | 2.5.6.25 | 2-339 |
| Completion Code - D37 | 2.5.6.26 | 2-340 |
| Completion Code - E37 | 2.5.6.27 | 2-341 |
| Compound (Complex) Conditions | 2.4.8.3.7 | 2-117 |
| COMPUTE Statement | 2.4.9.8 | 2-138 |
| Concept of Classes of Data | 2.3.4.4 | 2-7 |
| Concept of COBOL Levels | 2.3.4.2 | 2-6 |
| Concepts of Data Reference | 2.3.4 | 2-6 |
| Concepts of Top Down Structured Programing (TDSP) | 3.1.4 | 3-5 |
| Concepts of Top Down Structured Programing (TDSP), General | 3.1.4.1 | 3-5 |
| Conditional Expressions | 2.4.8.3.5 | 2-110 |
| Conditional Statements | 2.7.3.3 | 2-397 |
| Configuration Section | 2.4.6.2 | 2-28 |
| Console | 1.11.6.1 | 1-10 |
| Console Switches | 1.11.11 | 1-14 |
| Conspicuous Printing Style for Comment | 4.6.7 | 4-6 |
| Constant Count Indices | 4.9.6 | 4-8 |
| Continuation | 2.3.5.4 | 2-13 |
| Continuation of Lines | 2.3.5.4.1 | 2-13 |
| Continuation of Non-numeric Literals | 2.3.5.4.2 | 2-14 |
| Continuation of Numeric Literals | 2.3.5.4.3 | 2-14 |
| Control Block Pointers | 2.5.6.28 | 2-342 |
| Conventions, Language | 2.3 | 2-2 |
| COPY Statement | 2.4.9.9 | 2-142 |
| COPY Statement | 2.5.5.2 | 2-262 |
| Core Dump Tracing, DOS | 2.5.6.9 | 2-299 |
| Core Dump, OS/MVT | 2.5.6.29 | 2-348 |
| CPT, Chief Programer Team | 3.1.4.6 | 3-14 |
| Data Base Definition | 3.1.4.1.3 | 3-5 |
| DATA DESCRIPTION Clause | 2.4.7.13 | 2-75 |
| Data Description Entries | 2.3.5.3.2 | 2-12 |
| Data Division | 2.4.2.1.3 | 2-17 |
| Data Division | 2.4.4.1.3 | 2-20 |
| Data Division | 2.4.7 | 2-58 |
| Data Division Entries | 2.3.5.3 | 2-12 |
| Data Division Sort Feature | 2.5.3.3 | 2-241 |

| | Paragraph | Page |
|---|---|---|
| Data Flow Graph | 3.1.3.3 | 3-2 |
| Data Format Considerations | 2.7.1 | 2-383 |
| Data Formats | 2.9.4 | 2-421 |
| Data Item Considerations | 2.7.2 | 2-386 |
| Data Manipulation | 2.7.3.6 | 2-408 |
| DATA RECORD Clause | 2.4.7.11 | 2-74 |
| Data Variable Assignment | 4.9.3 | 4-8 |
| Data, Types of PDL Segments (Modules) | 3.1.4.3.4.4 | 3-10 |
| DATA-NAME or FILLER Clause | 2.4.7.14 | 2-77 |
| DATE-COMPILED Paragraph | 2.4.5.6 | 2-26 |
| DATE-WRITTEN Paragraph | 2.4.5.5 | 2-25 |
| Debugging Aids | 2.5.6 | 2-269 |
| Debugging Exercise, OC7 (Data Check) | 2.5.6.32 | 2-358 |
| Debugging of COBOL Segmented Programs Under OS/MFT | 2.5.6.33 | 2-368 |
| DEBUG-ITEM Special Register | 2.4.9.11 | 2-147 |
| Declaratives | 2.4.8.3.1 | 2-101 |
| Default Option | 2.4.3.3 | 2-18 |
| Definition of a Word | 2.3.3.1 | 2-2 |
| Definitions, USACSC Structured Programing Technology | 3.1.3 | 3-1 |
| DELETE Statement | 2.4.9.12 | 2-149 |
| Delimiters | 2.3.2.5 | 2-2 |
| Description, Program Design Language (PDL) or PSEUDO CODE | 3.1.4.3.2 | 3-7 |
| Descriptive Comments | 4.6.6 | 4-6 |
| Design Considerations | 4.2 | 4-1 |
| Design Walkthrough | 3.1.4.5.7.2 | 3-14 |
| Desk Checking | 4.7.2 | 4-7 |
| Detail Lines | 1.11.7.1.4 | 1-11 |
| Direct Access Storage Devices | 1.11.6.4 | 1-10 |
| DISPLAY Statement | 2.4.9.13 | 2-150 |
| Distribute Comment | 4.6.5 | 4-6 |
| DIVIDE Statement | 2.4.9.14 | 2-152 |
| Division Header | 2.3.5.2.2 | 2-11 |
| Divisions | 2.3.5.1.2 | 2-10 |
| Divisions, COBOL Program Structure | 2.4.4.1 | 2-20 |
| DO Statement | 2.4.9.15 | 2-155 |
| DO Statement | 2.5.1.1 | 2-227 |
| DO UNTIL Statement | 2.4.9.16 | 2-156 |
| DO UNTIL Statement | 2.5.1.4 | 2-232 |
| DO WHILE Statement | 2.4.9.16 | 2-157 |
| DO WHILE Statement | 2.5.1.2 | 2-228 |
| DOS COBOL Program Debugging Aids | 2.5.6.2 | 2-271 |
| DOS Core Dump Tracing | 2.5.6.9 | 2-299 |
| DOS, Commonly Encountered User Errors | 2.5.6.8 | 2-297 |

| | Paragraph | Page |
|---|---|---|
| Efficiency | 1.7.1.7 | 1-5 |
| Element Braces | 2.4.3.2.5 | 2-18 |
| Element, Language | 2.4.1.1 | 2-16 |
| Elements | 2.4.2.2 | 2-17 |
| Elements of File Design | 1.11.4 | 1-8 |
| Elements, Data Division | 2.4.7.1 | 2-58 |
| Elements, Environment Division | 2.4.6.1 | 2-27 |
| Elements, Identification Division | 2.4.5.1 | 2-23 |
| Ellipsis | 2.4.3.4 | 2-19 |
| ENTER Statement | 2.4.9.18 | 2-158 |
| Environment Division | 2.4.2.1.2 | 2-16 |
| Environment Division | 2.4.4.1.2 | 2-20 |
| Environment Division | 2.4.6 | 2-27 |
| Environment Division Sort Feature | 2.5.3.2 | 2-241 |
| Error Code | 4.8.2 | 4-8 |
| Error Condition Options | 1.11.10 | 1-14 |
| Errors, Commonly Encountered User | 2.5.6.8 | 2-297 |
| Example of Segmentation | 2.5.2.6 | 2-238 |
| Exception to the Use of ANSI COBOL | 1.4.3 | 1-3 |
| Exception to the Use of FORTRAN | 1.5.2 | 1-3 |
| Exception to the Use of SPEC | 1.4.2 | 1-2 |
| EXIT Statement | 2.4.9.19 | 2-159 |
| Figures | 3.1.4.2 | 3-6 |
| File Concept, Logical Record and | 2.3.4.1 | 2-6 |
| File Description (FD) and Sort-File (SD) Description Entries | 2.4.7.5 | 2-68 |
| File Design Considerations | 1.11.5 | 1-8 |
| File Organization | 1.11 | 1-7 |
| File Processing | 2.7.3.2 | 2-396 |
| FILE SECTION | 2.4.7.2 | 2-60 |
| FILE STATUS Clause | 2.4.6.15 | 2-51 |
| FILE-CONTROL Paragraph | 2.4.6.7 | 2-33 |
| FILLER Clause, DATA-NAME or | 2.4.7.14 | 2-77 |
| First Clause | 2.3.5.3.7 | 2-13 |
| First Header Line | 1.11.7.1.2 | 1-11 |
| Format | 1.11.8.1 | 1-13 |
| Format Presentation | 2.4.3.2 | 2-17 |
| Format Punctuation | 2.4.2 | 2-16 |
| Format Rules and Notes | 2.4.1 | 2-16 |
| Format, USACSC COBOL Specifications | 2.4.1.3 | 2-16 |
| Formats, COBOL Program Structure | 2.4.4.2 | 2-20 |
| FORTRAN | 1.5 | 1-3 |
| FORTRAN Character Set | 4.3.2 | 4-3 |
| FORTRAN Procedure | 1.5.1 | 1-3 |
| FORTRAN, Authority to Grant an Exception | 1.5.3 | 1-4 |
| FORTRAN, Exception to the Use of | 1.5.2 | 1-3 |
| Function Brackets | 2.4.3.2.6 | 2-18 |
| Function, USACSC COBOL Specifications | 2.4.1.2 | 2-16 |
| Functional Breakdown | 3.1.4.1.1 | 3-5 |

| | Paragraph | Page |
|---|---|---|
| General Description of Reference Format | 2.3.5.1 | 2-10 |
| General Description, Format Punctuation | 2.4.2.1 | 2-16 |
| General Description, Procedure Division | 2.4.8.1 | 2-99 |
| General Format | 3.1.7.1.4.1 | 3-19 |
| General Rules, Procedure Division | 2.4.8.3 | 2-100 |
| General Rules, USACSC COBOL Specifications | 2.4.1.5 | 2-16 |
| General Standards and Guidelines | 3.1.5 | 3-16 |
| General, Concepts of Top Down Structured Programing (TDSP) | 3.1.4.1 | 3-5 |
| General, Programing Support Library (PSL) | 3.1.4.4.1 | 3-11 |
| General, Symbols and Notations Used in This Manual | 2.4.3.1 | 2-17 |
| General, USACSC Programing Procedures | 1.1 | 1-1 |
| General, USACSC Structured Programing Technology | 3.1.1 | 3-1 |
| Glossary | 2.3.1 | 2-2 |
| GO TO Statement | 2.4.9.20 | 2-160 |
| Grouping of Arguments | 4.8.1 | 4-8 |
| Guidelines | 3.1.5.2 | 3-17 |
| Guidelines for Semantics | 3.1.4.3.6 | 3-10 |
| Halts | 1.11.8.5 | 1-13 |
| Header, Division | 2.3.5.2.2 | 2-11 |
| Hit Ratio | 1.11.5.2 | 1-9 |
| Identification Division | 2.4.2.1.1 | 2-16 |
| Identification Division | 2.4.4.1.1 | 2-20 |
| Identification Division | 2.4.5 | 2-23 |
| Identification of Program in a Comment | 4.6.2 | 4-5 |
| IF Statement | 2.4.9.21 | 2-161 |
| IF Statement | 2.5.1.3 | 2-230 |
| Implementing Instructions, Single Source Library System | 2.8.5 | 2-419 |
| Include Capability | 3.1.7 | 3-17 |
| Included, Types of PDL Segments (Modules) | 3.1.4.3.4.3 | 3-10 |
| Indentation and Formatting Conventions | 3.1.7.1.4 | 3-19 |
| Indexed File Organization | 1.11.2 | 1-7 |
| Indexing | 2.3.4.8 | 2-9 |
| Indexing | 2.5.4.4 | 2-249 |
| Input Buffers | 2.9.2.1 | 2-420 |
| Input Media | 1.11.6 | 1-10 |
| Input/Output Devices | 4.9.5 | 4-8 |
| Input/Output Functions | 4.2.3 | 4-1 |
| Input/Storage Areas | 2.9.2 | 2-420 |
| INPUT-OUTPUT Section | 2.4.6.6 | 2-32 |
| INSPECT Statement | 2.4.9.22 | 2-166 |

| | Paragraph | Page |
|---|---|---|
| INSTALLATION Paragraph | 2.4.5.4 | 2-25 |
| Integration Testing | 3.1.3.12.2 | 3-4 |
| Interacting Factors | 1.11.5.1 | 1-8 |
| Internal Module Structure | 3.1.4.3.5 | 3-10 |
| Interpretating Output | 2.5.6.10 | 2-309 |
| Introduction to Copy Library Facility | 2.5.5.1 | 2-261 |
| Introduction to Debugging Aids | 2.5.6.1 | 2-269 |
| Introduction, FORTRAN Programing Procedures | 4.1 | 4-1 |
| Introduction, Sort Feature | 2.5.3.1 | 2-240 |
| Introduction, Table Handling Feature | 2.5.4.1 | 2-242 |
| Introduction, Programing Procedures | 1.2 | 1-1 |
| Introduction, USACSC Standard Portable Expanded COBOL, ANSI COBOL Subset | 2.2 | 2-1 |
| Introduction, USACSC Structured Programing Technology | 3.1 | 3-1 |
| INVALID KEY Option | 2.9.9 | 2-422 |
| IPO (Input, Process, Output) Chart | 3.1.3.4 | 3-2 |
| I-O CONTROL Paragraph | 2.4.6.16 | 2-52 |
| JUSTIFIED Clause | 2.4.7.21 | 2-95 |
| KEY Option, INVALID | 2.9.9 | 2-422 |
| LABEL RECORDS Clause | 2.4.7.6 | 2-69 |
| Language Conventions | 2.3 | 2-2 |
| Language Element | 2.4.1.1 | 2-16 |
| Language Structure | 2.3.2 | 2-2 |
| Level Indicators | 2.3.5.3.1 | 2-12 |
| Level-Number 77 | 2.3.5.3.4 | 2-13 |
| Level-Number 88 | 2.3.5.3.5 | 2-13 |
| Level-Numbers | 2.3.4.3 | 2-7 |
| Level-Numbers Ø1 Through 49 | 2.3.5.3.3 | 2-13 |
| Librarian | 3.1.3.5 | 3-2 |
| Librarian | 3.1.4.6.1 | 3-15 |
| Library Facility, Source Program | 2.5.5 | 2-261 |
| Library Functions | 4.2.2 | 4-1 |
| Link Edit Map | 2.5.6.4 | 2-273 |
| LINKAGE SECTION | 2.4.7.4 | 2-66 |
| Logical Flow | 1.7.1.5 | 1-5 |
| Logical Operators | 4.3.3.3 | 4-4 |
| Logical Record and File Concept | 2.3.4.1 | 2-6 |
| Lower-Case Words | 2.4.3.2.4 | 2-17 |

| | Paragraph | Page |
|---|---|---|
| Machine Independence | 1.7.1.2 | 1-5 |
| Main, Types of PDL Segments (Modules) | 3.1.4.3.4.1 | 3-10 |
| Maintainability | 1.7.1.3 | 1-5 |
| Meaningful Comments | 4.6.1 | 4-5 |
| Message Number | 1.11.8.3 | 1-13 |
| Misleading Rules of File Design | 1.11.5.3 | 1-9 |
| Modular Structure | 3.1.4.1.2 | 3-5 |
| Modularity | 4.2.1 | 4-1 |
| Module Delimitation | 3.1.4.3.3 | 3-10 |
| Module Size | 3.1.7.1.6 | 3-20 |
| MOVE Statement | 2.4.9.23 | 2-174 |
| MULTIPLE FILE TAPE Clause | 2.4.6.19 | 2-57 |
| Multiple Program Outputs | 1.10 | 1-6 |
| MULTIPLY Statement | 2.4.9.24 | 2-179 |
| Nested IF | 3.1.7.1.5 | 3-20 |
| Nine Step Module Management Process | 3.1.4.7 | 3-15 |
| Non-integer Variable | 4.9 | 4-8 |
| Object Storage Layout | 2.5.6.5 | 2-274 |
| OBJECT-COMPUTER Paragraph | 2.4.6.4 | 2-29 |
| Objective, Single Source Library System | 2.8.1 | 2-418 |
| Objectives | 1.3 | 1-2 |
| OCCURS Clause | 2.4.7.17 | 2-81 |
| OCCURS Statement | 2.5.4.3 | 2-247 |
| OC7 (Data Check) Debugging Exercise | 2.5.6.32 | 2-358 |
| OCx Completion Code Note | 2.5.6.17 | 2-331 |
| OPEN Statement | 2.4.9.25 | 2-181 |
| Operators Used in FORTRAN Programs | 4.3.3 | 4-4 |
| Option, Default | 2.4.3.3 | 2-18 |
| Option, INVALID KEY | 2.9.9 | 2-422 |
| Organization of Segmentation Facility | 2.5.2.1 | 2-235 |
| OS COBOL Program Debugging Aids | 2.5.6.11 | 2-326 |
| OS Data Exceptions, Recognition and Error Recovery | 2.5.6.30 | 2-356 |
| OS/DOS Compatibility | 2.9 | 2-420 |
| OS/MVT Core Dump | 2.5.6.29 | 2-348 |
| Output Media | 1.11.7 | 1-10 |
| Overlay Structures | 2.7.5.1 | 2-414 |
| Page Line | 1.11.7.1.5 | 1-11 |
| Paragraph Naming | 2.7.3.1 | 2-396 |
| Paragraph-Name and Paragraph | 2.3.5.2.4 | 2-12 |
| Paragraph/Section Names | 2.3.6.1.4 | 2-15 |
| PDL Example | 3.1.4.3.8 | 3-11 |
| PDL Segments (Modules), Types of | 3.1.4.3.4 | 3-10 |
| PERFORM Statement | 2.4.9.26 | 2-185 |

| | Paragraph | Page |
|---|---|---|
| PICTURE Clause | 2.4.7.18 | 2-85 |
| PICTURE Clause | 2.9.7 | 2-422 |
| Printer | 1.11.7.1 | 1-11 |
| Printer, Detail Lines | 1.11.7.1.4 | 1-11 |
| Printer, First Header Line | 1.11.7.1.2 | 1-11 |
| Printer, Page Line | 1.11.7.1.5 | 1-11 |
| Printer, Remaining Header Lines | 1.11.7.1.3 | 1-11 |
| Printer, Security Classification | 1.11.7.1.1 | 1-11 |
| Printer, Skipping | 1.11.7.1.7 | 1-12 |
| Printer, Spacing | 1.11.7.1.6 | 1-12 |
| Procedure Division | 2.4.2.1.4 | 2-17 |
| Procedure Division | 2.4.4.1.4 | 2-20 |
| Procedure Division | 2.4.8 | 2-99 |
| Procedure Division Considerations for Table Handling | 2.5.4.5 | 2-251 |
| Procedure Division Design | 2.6.1 | 2-372 |
| Procedure Division Sort Feature | 2.5.3.4 | 2-242 |
| Procedure Division Techniques | 2.7.3 | 2-396 |
| Procedures, Single Source Library System | 2.8.2 | 2-419 |
| Productivity | 1.7.1.4 | 1-5 |
| Program Comments | 4.6 | 4-5 |
| Program Comments for Subroutines | 4.6.4 | 4-6 |
| Program Design Criteria | 1.8 | 1-6 |
| Program Design Language (PDL) or PSEUDO CODE | 3.1.3.6 | 3-2 |
| Program Design Language (PDL) or PSEUDO CODE | 3.1.4.3 | 3-6 |
| Program Design Language (PDL) or PSEUDO CODE, Description | 3.1.4.3.2 | 3-7 |
| Program Design Techniques, USACSC COBOL | 2.6 | 2-372 |
| Program Identification | 1.9 | 1-6 |
| Program Level | 3.1.3.8.2 | 3-3 |
| Program Logic Checklist | 4.7.3 | 4-7 |
| Program Modification | 4.6.3 | 4-6 |
| Program Organization | 3.1.7.1.2 | 3-18 |
| Program Segments, Structure of | 2.5.2.4 | 2-236 |
| Program Structure | 4.3 | 4-1 |
| Program Structure Techniques, COBOL | 2.7 | 2-381 |
| Program Switches | 2.9.6 | 2-421 |
| Program Techniques, OS/DOS Compatibility | 2.9.1 | 2-420 |
| Program to Operator Messages | 1.11.8 | 1-12 |
| Program, Structured | 3.1.3.9 | 3-3 |
| PROGRAM-ID | 1.11.8.2 | 1-13 |
| PROGRAM-ID Paragraph | 2.4.5.2 | 2-24 |
| Programer, Backup | 3.1.3.1 | 3-1 |
| Programer, Chief | 3.1.3.2 | 3-2 |
| Programing Support Library (PSL) | 3.1.3.7 | 3-2 |
| Programing Support Library (PSL) | 3.1.4.4 | 3-11 |
| Programing, USACSC Concepts | 1.7 | 1-4 |

| | Paragraph | Page |
|---|---|---|
| PSEUDO CODE, Program Design Language (PDL) or | 3.1.3.6 | 3-2 |
| PSEUDO CODE, Program Design Language (PDL) or | 3.1.4.3 | 3-6 |
| PSL, Programing Support Library | 3.1.3.7 | 3-2 |
| PSL, Programing Support Library | 3.1.4.4 | 3-11 |
| Punch | 1.11.7.2 | 1-12 |
| Punctuation Characters | 2.3.2.2 | 2-2 |
| Purpose of USACSC Standard Portable Expanded COBOL, ANSI COBOL Subset Specifications | 2.2.1 | 2-1 |
| Purpose, Program Design Language (PDL) or PSEUDO CODE | 3.1.4.3.1 | 3-6 |
| Purpose, USACSC Structured Programing Technology | 3.1.2 | 3-1 |
| Qualification of Name | 2.3.4.6 | 2-8 |
| Quotation Marks | 2.3.2.3 | 2-2 |
| Random File Organization | 1.11.3 | 1-8 |
| READ Statement | 2.4.9.27 | 2-195 |
| Readability | 2.3.6.1.8 | 2-15 |
| Real and Integer Data | 4.2.4 | 4-1 |
| RECORD CONTAINS Clause | 2.4.7.7 | 2-70 |
| RECORD DESCRIPTION Clause | 2.4.7.12 | 2-74 |
| Record Identifier | 2.9.5 | 2-421 |
| RECORD KEY Clause | 2.4.6.13 | 2-50 |
| Recovery Guidance | 1.11.9 | 1-14 |
| Recovery Procedures In Comments | 4.6.8 | 4-6 |
| REDEFINES Clause | 2.4.7.15 | 2-78 |
| Reference Format Representation | 2.3.5.2 | 2-10 |
| Reference Format, Clause | 2.3.5.3.6 | 2-13 |
| Reference Format, Clauses | 2.3.5.3.8 | 2-13 |
| Reference Format, Data Description Entries | 2.3.5.3.2 | 2-12 |
| Reference Format, Data Division Entries | 2.3.5.3 | 2-12 |
| Reference Format, Division Header | 2.3.5.2.2 | 2-11 |
| Reference Format, Divisions | 2.3.5.1.2 | 2-10 |
| Reference Format, First Clause | 2.3.5.3.7 | 2-13 |
| Reference Format, General Description of | 2.3.5.1 | 2-10 |
| Reference Format, Level Indicators | 2.3.5.3.1 | 2-12 |
| Reference Format, Level-Numbers Ø1 Through 49 | 2.3.5.3.3 | 2-13 |
| Reference Format, Level-Number 77 | 2.3.5.3.4 | 2-13 |
| Reference Format, Level-Number 88 | 2.3.5.3.5 | 2-13 |
| Reference Format, Paragraph-Name and Paragraph | 2.3.5.2.4 | 2-12 |
| Reference Format, Rules | 2.3.5.1.1 | 2-10 |
| Reference Format, Section Header | 2.3.5.2.3 | 2-12 |
| Reference Format, Sequence Numbers | 2.3.5.2.1 | 2-11 |
| Reference Format, USACSC COBOL | 2.3.5 | 2-10 |

| | Paragraph | Page |
|---|---|---|
| Register and Save Area | 2.5.6.31 | 2-357 |
| Relation Operators | 4.3.3.2 | 4-4 |
| RELATIVE KEY Clause | 2.4.6.12 | 2-48 |
| RELEASE Statement | 2.4.9.28 | 2-198 |
| Remaining Header Lines | 1.11.7.1.3 | 1-11 |
| REMARKS Paragraph | 2.4.5.8 | 2-27 |
| RERUN Clause | 2.4.6.17 | 2-55 |
| RESERVE Clause | 2.4.6.10 | 2-47 |
| Reserved Words | 2.3.3.2.1 | 2-3 |
| Restricted COBOL Statement Usage | 3.1.7.1.1 | 3-18 |
| RETURN Statement | 2.4.9.29 | 2-199 |
| REWRITE Statement | 2.4.9.30 | 2-200 |
| Rules | 2.3.5.1.1 | 2-10 |
| Rules, Ellipsis | 2.4.3.4.1 | 2-19 |
| Rules, Reference Format | 2.3.5.1.1 | 2-10 |
| SAME Clause | 2.4.6.18 | 2-56 |
| SEARCH Statement | 2.4.9.31 | 2-202 |
| Section Header | 2.3.5.2.3 | 2-12 |
| Security Classification | 1.11.7.1.1 | 1-11 |
| SECURITY Paragraph | 2.4.5.7 | 2-26 |
| Segment Classification | 2.5.2.2 | 2-236 |
| Segmentation Control | 2.5.2.3 | 2-236 |
| Segmentation, Example of | 2.5.2.6 | 2-238 |
| Segmentation, USACSC Guidelines for | 2.5.2.7 | 2-239 |
| SELECT Clause | 2.4.6.8 | 2-34 |
| Self-documenting Programs | 1.7.1.1 | 1-5 |
| Separators | 2.3.2.1 | 2-2 |
| Sequence Numbers | 2.3.5.2.1 | 2-11 |
| Sequence Numbers | 4.3.1.2 | 4-2 |
| Sequential File Organization | 1.11.1 | 1-7 |
| SET Statement | 2.4.9.32 | 2-203 |
| SIGN Clause | 2.4.7.16 | 2-79 |
| Simple Conditions | 2.4.8.3.6 | 2-110 |
| Simplistic Approach | 1.7.1 | 1-4 |
| Single Source Library System | 2.8 | 2-418 |
| Single Source Library System, Coding | 2.8.3 | 2-419 |
| Single Source Library System, Implementing Instructions | 2.8.5 | 2-419 |
| Single Source Library System, Objective | 2.8.1 | 2-418 |
| Single Source System | 2.8.4 | 2-419 |
| Skipping | 1.11.7.1.7 | 1-12 |

| | Paragraph | Page |
|---|---|---|
| SORT Feature | 2.5.3 | 2-240 |
| SORT Feature, Data Division | 2.5.3.3 | 2-241 |
| SORT Feature, Environment Division | 2.5.3.2 | 2-241 |
| SORT Feature, Procedure Division | 2.5.3.4 | 2-242 |
| SORT Statement | 2.4.9.33 | 2-204 |
| Source Card Coding | 4.3.1 | 4-1 |
| Source Language System (SLS)/Program Language Update Service (PLUS) | 2.7.6 | 2-418 |
| Source Library Maintenance | 2.7.6.1 | 2-418 |
| Source Program Library Facility | 2.5.5 | 2-261 |
| SOURCE-COMPUTER Paragraph | 2.4.6.3 | 2-28 |
| Spacing | 1.11.7.1.6 | 1-12 |
| SPEC | 1.4 | 1-2 |
| SPEC Procedures | 1.4.1 | 1-2 |
| Special Features | 2.5 | 2-226 |
| SPECIAL-NAMES Paragraph | 2.4.6.5 | 2-30 |
| Specification Walkthrough | 3.1.4.5.7 | 3-14 |
| Standard Coding Conventions, USACSC | 2.3.6 | 2-14 |
| Standard Construct | 1.7.1.6 | 1-5 |
| Standard Logic Constructs | 2.6.2 | 2-373 |
| Standards | 3.1.5.1 | 3-16 |
| START Statement | 2.4.9.34 | 2-213 |
| Statement Labeling | 4.3.1.3 | 4-2 |
| Statement Number | 4.7.3.1 | 4-7 |
| Statement Ordering | 4.3.1.4 | 4-3 |
| Statements | 2.4.8.3.2 | 2-101 |
| Statements | 2.4.9 | 2-121 |
| Statements, Branching | 2.7.3.5 | 2-405 |
| Step 1 Module Identified | 3.1.4.7.1 | 3-15 |
| Step 2 Module Documented | 3.1.4.7.2 | 3-15 |
| Step 3 Joint Review of Module | 3.1.4.7.3 | 3-15 |
| Step 4 Module Accepted | 3.1.4.7.4 | 3-15 |
| Step 5 Module Coded and Compiled | 3.1.4.7.5 | 3-15 |
| Step 6 Module Test Plan Complete | 3.1.4.7.6 | 3-15 |
| Step 7 Module Quality Assurance Review | 3.1.4.7.7 | 3-16 |
| Step 8 Module Linked | 3.1.4.7.8 | 3-16 |
| Step 9 Module Ready | 3.1.4.7.9 | 3-16 |
| STOP RUN Statement | 2.9.3 | 2-420 |
| STOP Statement | 2.4.9.35 | 2-214 |
| Structure Chart | 3.1.3.8 | 3-2 |
| Structure of Program Segments | 2.5.2.4 | 2-236 |
| Structure of the COBOL Program | 2.4.4.3 | 2-2 |

| | Paragraph | Page |
|---|---|---|
| Structured Program | 3.1.3.9 | 3-3 |
| Structured Programing (SP) or Structured Coding (SC) | 3.1.3.10 | 3-3 |
| Structured Programing Statements - MetaCOBOL Macro Facility | 2.5.1 | 2-226 |
| Structured Source Code Listing | 3.1.3.11 | 3-3 |
| Structured Testing | 3.1.3.12 | 3-4 |
| Structured Walkthrough | 3.1.3.13 | 3-4 |
| Structured Walkthroughs | 3.1.4.5 | 3-13 |
| Structures | 3.1.7.1.4.2 | 3-20 |
| Structure, Language | 2.3.2 | 2-2 |
| Structure, Procedure Division | 2.4.8.2 | 2-99 |
| Stub | 3.1.3.14 | 3-4 |
| Subprogram Linkage | 2.7.5.2 | 2-415 |
| Subprogram Technique | 2.7.5.3 | 2-416 |
| Subroutine, Types of PDL Segments (Modules) | 3.1.4.3.4.2 | 3-10 |
| Subscripted Variables | 4.7.3.4 | 4-7 |
| Subscripting | 2.3.4.7 | 2-9 |
| Subscripting | 2.5.4.2 | 2-243 |
| SUBTRACT Statement | 2.4.9.36 | 2-216 |
| Support Members | 3.1.4.6.2 | 3-15 |
| Switches, Console | 1.11.11 | 1-14 |
| Switches, Program | 2.9.6 | 2-421 |
| Symbolic Names | 4.3.1.5 | 4-3 |
| Symbols and Notations Used in This Manual | 2.4.3 | 2-17 |
| Symbols/Words | 2.3.6.1.1 | 2-15 |
| SYNCHRONIZED Clause | 2.4.7.22 | 2-96 |
| Syntax Errors | 2.9.10 | 2-422 |
| Syntax Rules | 2.4.1.4 | 2-16 |
| System Action Under Cancel | 2.5.6.6 | 2-285 |
| System Level | 3.1.3.8.1 | 3-2 |
| Systems Testing | 3.1.3.12.3 | 3-4 |
| Table Construction and Referencing | 2.7.4.1 | 2-408 |
| Table Handling Feature | 2.5.4 | 2-242 |
| Table Handling Techniques | 2.7.4 | 2-408 |
| Table Handling, Procedure Division Considerations | 2.5.4.5 | 2-251 |
| Tape | 1.11.6.2 | 1-10 |
| Tapes and Direct Access Storage Devices | 1.11.7.3 | 1-12 |
| Team Operation or Chief Programer Team | 3.1.3.15 | 3-4 |
| Test Walkthrough | 3.1.4.5.7.4 | 3-14 |
| Top Down Development | 3.1.3.16 | 3-4 |
| Top Down Development | 3.1.4.1.4 | 3-6 |
| Top Down Program (TDP) | 3.1.3.17 | 3-5 |
| Top Down Structured Programing (TDSP) | 3.1.3.18 | 3-5 |
| Top Down Structured Programing (TDSP), Concepts | 3.1.4 | 3-5 |

| | Paragraph | Page |
|---|---|---|
| Tracing, DOS Core Dump | 2.5.6.9 | 2-299 |
| Transfer of Control | 2.7.5 | 2-414 |
| Type of Message | 1.11.8.4 | 1-13 |
| Types of PDL Segments (Modules) | 3.1.4.3.4 | 3-10 |
| Types of Words | 2.3.3.2 | 2-2 |
| Unit Testing | 3.1.3.12.1 | 3-4 |
| Upper-Case Words (Underlined) | 2.4.3.2.2 | 2-17 |
| Upper-Case Words (Underlined/Not Underlined) | 2.4.3.2.3 | 2-17 |
| USACSC COBOL Program Design Techniques | 2.6 | 2-372 |
| USACSC COBOL Reference Format | 2.3.5 | 2-10 |
| USACSC COBOL Specifications | 2.4 | 2-16 |
| USACSC Guidelines | 2.4.1.6 | 2-16 |
| USACSC Guidelines for Segmentation | 2.5.2.7 | 2-239 |
| USACSC Program Design Language (PDL) Conventions | 3.1.4.3.7 | 3-11 |
| USACSC Programing Concepts | 1.7 | 1-4 |
| USACSC Programing Procedures, General | 1.1 | 1-1 |
| USACSC SPEC COBOL Language Standards and Guidelines | 3.1.6 | 3-17 |
| USACSC Standard Coding Conventions | 2.3.6 | 2-14 |
| USACSC Structured Programing Technology, Definitions | 3.1.3 | 3-1 |
| USACSC Structured Programing Technology, General | 3.1.1 | 3-1 |
| USACSC Structured Programing Technology, Introduction | 3.1 | 3-1 |
| USACSC Structured Programing Technology, Purpose | 3.1.2 | 3-1 |
| USAGE Clause | 2.4.7.19 | 2-92 |
| USE FOR DEBUGGING Statement | 2.4.9.10 | 2-145 |
| USE Statement | 2.4.9.37 | 2-219 |
| User Errors, Commonly Encountered | 2.5.6.8 | 2-297 |
| User-Defined Words | 2.3.3.2.2 | 2-5 |
| Utility Programs and Subroutines | 1.11.12 | 1-14 |
| VALUE Clause | 2.4.7.20 | 2-93 |
| VALUE OF Clause | 2.4.7.8 | 2-70 |
| Variable Names | 4.5 | 4-5 |
| Verbs | 2.3.6.1.2 | 2-15 |
| Verify Statement Number | 4.7.3.2 | 4-7 |
| Wait States | 2.5.6.7 | 2-296 |
| Whole Numbers | 4.9.4 | 4-8 |
| Word, Definition | 2.3.3.1 | 2-2 |
| Words | 2.3.3 | 2-2 |
| Words | 2.4.3.2.1 | 2-17 |
| Words, Lower-Case | 2.4.3.2.4 | 2-17 |
| Words, Reserved | 2.3.3.2.1 | 2-3 |
| Words, Types | 2.3.3.2 | 2-2 |
| Words, Upper-Case (Underlined) | 2.4.3.2.2 | 2-17 |
| Words, Upper-Case (Underlined/Not Underlined) | 2.4.3.2.3 | 2-17 |
| Words, User-Defined | 2.3.3.2.2 | 2-5 |
| Working-Storage Section | 2.3.6.1.3 | 2-15 |
| Working-Storage Section | 2.4.7.3 | 2-62 |
| WRITE Statement | 2.4.9.38 | 2-221 |

DA
FILM