

Licensed Material - Property of IBM

LY28-6486-2

**Program Product**

**IBM OS/VS COBOL Compiler  
Program Logic**

**Program Number 5740-CB1**

**IBM**

Second Edition (November 1976)

This edition replaces the previous edition (numbered LY28-6486-0) and its technical newsletter (numbered LN20-9106) and makes them obsolete.

This edition corresponds to the second release of the IBM OS/VS COBOL Compiler and to any subsequent releases unless otherwise indicated in new editions or technical newsletters.

Technical changes are summarized under "Summary of Amendments" following the list of figures. Each technical change is marked by a vertical line to the left of the change. In addition, miscellaneous editorial and technical changes have been made throughout the publication.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using the publication, consult the latest IBM System/370 Bibliography, GC20-0001, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

Forms for readers' comments are provided at the back of the publication. If the forms have been removed, comments may be addressed to IBM Corporation, P. O. Box 50020, Programming Publishing, San Jose, California 95150. All comments and suggestions become the property of IBM.

---

**Summary of Amendments**

**Number 1**

---

*Date of Publication:* November 1, 1976

*Form of Publication:* Revision, LY28-6486-2

**OS/VS COBOL RELEASE 2**

*New:* Programming Feature

The text has been updated to reflect Release 2 of the OS/VS COBOL Compiler. Major updates include:

- Phase 04, a new phase that processes the COPY statement and performs BASIS processing.
- Phase 35, a new phase that processes USE FOR DEBUGGING statements and their operands.
- A new compiler option, LANGLVL, applies to those language elements where the interpretation differs from the 1968 American National Standard, X3.23-1968, COBOL, to the 1974 American National Standard, X3.23-1974, COBOL.
- 
- Phases 05, 06, 08, and 80 flowcharts have been added.
- Extensive updates have been made to; Communications Area, Compiler Table Formats, and Internal Text Formats in the Data Areas Section.
- Several updates, including a list of user ABEND codes, have been made to the Diagnostic Section.

**Miscellaneous Changes**

*Maintenance:* Documentation Only

Some maintenance changes have also been made.



## PREFACE

This publication describes the internal design of the IBM OS/VS COBOL Release 2 compiler. The manual is intended for use by persons involved in program support and by systems programmers involved in altering the program design for installations requiring such alteration. It supplements the compiler listing and its comments but is not a substitute for them.

The publication is divided into the following sections:

- An introduction that describes the compiler functions and specifies the relationship of the compiler to the operating system.
- A Method of Operations section that includes a section on each of the compiler phases. Within these chapters, the material is organized by phase functions and is not necessarily presented in the order in which the phase operations are performed.
- A Program Organization section that includes one chart of the overall logic flow for each phase and other charts showing detailed descriptions of some of the major phase routines.
- A Directory section that shows register usage, flowchart labels, and the tables used by each phase.
- A Data Areas section showing the formats of the compiler Communication Region, the dictionary, the internal texts, the tables that occupy the table-handling area, and the tables that are created for object-time debugging purposes.
- A Diagnostic Aids section.
- An appendix that describes the routines that handle compiler tables and the dictionary.
- An appendix that describes the object module produced by the compiler.
- An appendix that describes the Report Writer subprogram.
- An appendix that describes the interface with the Conversational Monitor System (CMS).

- A Glossary of special terms.
- Foldout Diagrams.
- An index.

Effective use of this manual requires an extensive knowledge of the IBM Assembler Language, OS/VS System Control, and the IBM OS/VS COBOL language. Prerequisite and related publications include:

IBM OS/VS Supervisor Services and Macro Instructions, Order No. GC28-0683

IBM OS/VS Data Management Services Guide, Order No. GC26-3783

IBM OS/VS Data Management Macro Instructions, Order No. GC26-3793

IBM OS/VS Linkage Editor and Loader, Order No. GC26-3813

IBM OS/VS and DOS/VS Assembler Language Guide, Order No. GC33-4010

IBM OS/VS Services Aids, Order No. GC28-0633

IBM OS/VS System Utilities, Order No. GC35-0005

IBM OS/VS System Management Facilities (SMF), Order No. GC35-0004

IBM OS/VS Data Management for System Programmers, Order No. GC26-3837

IBM OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide, Order No. GC26-3838

Prerequisite Program Product documents include:

IBM VS COBOL for OS/VS, Order No. GC26-3857

IBM OS/VS COBOL Compiler and Library Programmer's Guide, Order No. SC28-6483

IBM OS/VS COBOL Compiler and Library Installation Reference Material, Order No. SC28-6481

IBM OS/VS COBOL Subroutine Library Program Logic, Order No. LY28-6425

ACKNOWLEDGMENT

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

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

"COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

"No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

"The authors and copyright holders of the copyrighted material used herein

FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the UNIVAC (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications."

Table of contents

SECTION 1. INTRODUCTION . . . . .	19	Processing for the BATCH Option . . . . .	48
Relationship of the Compiler to the IBM OS/VS System . . . . .	19	Entering Statistical Information In COMMON . . . . .	48
Characteristics of the Compiler . . . . .	19	Preparing For Federal Information Processing Standard (FIPS) Flagging . . . . .	48
Physical Characteristics . . . . .	19	Information Returned to Phase 00 . . . . .	49
Operational Characteristics . . . . .	20	Error Conditions . . . . .	49
Input . . . . .	20		
Output . . . . .	20	PHASE 03 . . . . .	50
Design of the Compiler . . . . .	20	Obtaining and Printing Error Messages . . . . .	50
Compiler Data Sets . . . . .	21	Returning Control to Phase 00 . . . . .	50
Compiler Phases . . . . .	21		
Phase 00 (IKFCBL00) . . . . .	21	PHASE 04 . . . . .	51
Phase 01 (IKFCBL01) . . . . .	22	Input . . . . .	51
Phase 02 (IKFCBL02) . . . . .	22	Output . . . . .	51
Phase 03 (IKFCBL03) . . . . .	22	Error Conditions . . . . .	51
Phase 04 (IKFCBL04) . . . . .	22	Tables Used . . . . .	51
Phase 05 (IKFCBL05) . . . . .	22	Main Flow of Control in PHASE 04 . . . . .	51
Phase 06 (IKFCBL06) . . . . .	22	Processing Routines Used . . . . .	51
Phase 08 (IKFCBL08) . . . . .	22		
Phase 10 (IKFCBL10) . . . . .	22	PHASE 05 . . . . .	52
Phase 12 (IKFCBL12) . . . . .	22	Input . . . . .	52
Phase 1B (IKFCBL1B) . . . . .	22	Output . . . . .	52
Phase 20 (IKFCBL20) . . . . .	23	Syntax Language Summary . . . . .	52
Phase 22 (IKFCBL22) . . . . .	23	Error Conditions . . . . .	52
Phase 21 (IKFCBL21) . . . . .	23		
Phase 25 (IKFCBL25) . . . . .	23	PHASE 06 . . . . .	54
Phase 3 (IKFCBL30) . . . . .	23	Input . . . . .	54
Phase 35 (IKFCBL35) . . . . .	23	Output . . . . .	54
Phase 4 (IKFCBL40) . . . . .	23		
Phase 45 (IKFCBL45) . . . . .	23	PHASE 08 . . . . .	55
Phase 50 (IKFCBL50) . . . . .	24	Input . . . . .	55
Phase 51 (IKFCBL51) . . . . .	24	Output . . . . .	55
Phase 6 (IKFCBL60) . . . . .	24	Processing . . . . .	55
Phases 62, 63, and 64 (IKFCBL62, IKFCBL63, and IKFCBL64) . . . . .	24	Error Conditions . . . . .	55
Phase 65 (IKFCBL65) . . . . .	24	PHASE 10 . . . . .	56
Phase 6A (IKFCBL6A) . . . . .	24	Major Working Routines . . . . .	56
Phases 70, 71, and 72 (IKFCBL70, IKFCBL71, and IKFCBL72) . . . . .	25	GETDLM Routine . . . . .	56
Phase 80 (IKFCBL80) . . . . .	25	GETWD Routine . . . . .	56
Compiler Options . . . . .	25	GETCRD Routine . . . . .	57
Storage Requirements . . . . .	29	Identification Division . . . . .	57
		Environment Division . . . . .	57
		Configuration Section . . . . .	57
		Input-Output Section . . . . .	58
		FILE-CONTROL Paragraph . . . . .	58
		I-O-CONTROL Paragraph . . . . .	58
		Data Division . . . . .	59
		File Section . . . . .	59
		File Description Entries . . . . .	59
		Sort Description Entries . . . . .	59
		Record Description Entries . . . . .	60
		Working-Storage and Linkage Sections . . . . .	60
		Communication Section . . . . .	60
		PHASE 12 . . . . .	61
		Operations in Other Phases . . . . .	61
		REPORT Clause . . . . .	61
		Report Section Header . . . . .	61
		USE Sentences . . . . .	61
		Procedure Division Verbs . . . . .	61
		Control-Field Save-Area Names . . . . .	62
		REDEFINES Clause . . . . .	62
		Producing the Report Writer Subprogram (RWS) . . . . .	62
		Routine RDSCAN . . . . .	62
		Routine PROC01 . . . . .	64
SECTION 2. METHOD OF OPERATION . . . . .	31		
Phase 00 . . . . .	31		
Receiving Control from the Operating System . . . . .	31		
Returning Control to the Operating System . . . . .	31		
Processing Between Phases . . . . .	31		
Phase Input/Output Requests . . . . .	34		
Compiler Linkage to Data Management . . . . .	44		
Directing Error Messages and Progress Messages to the SYSTEM Data Set . . . . .	44		
Segmentation Operations . . . . .	44		
Table and Dictionary Handling . . . . .	44		
Communications Area (COMMON) . . . . .	44		
Syntax-Checking Compilations . . . . .	45		
Terminal Error Conditions . . . . .	45		
PHASE 01 . . . . .	46		
Compilation Parameters . . . . .	46		
Return From Phase 02 . . . . .	46		
PHASE 02 . . . . .	47		
Compilation Parameters . . . . .	47		
Buffer Size Determination . . . . .	47		

Routine PROC02 . . . . .	64	Statements with CORRESPONDING	
Routine FLUSH . . . . .	64	Options . . . . .	89
Routine GNSPRT . . . . .	64	SEARCH Verb Strings . . . . .	91
Generating Error Messages . . . . .	64	Determining the Uniqueness of a	
Generating the Source Listing . . . . .	65	Name . . . . .	95
Information for Later Phases . . . . .	65	Replacing Names with Dictionary	
		Attributes . . . . .	95
PHASE 1B . . . . .	66	Debugging . . . . .	95
Procedure-names . . . . .	66	Error Processing . . . . .	96
Entering Procedure-names in the			
Dictionary . . . . .	67	PHASE 35 . . . . .	99
Using the PNTABL and PNOTBL Tables . . . . .	67	Table Handling . . . . .	100
Storing Information . . . . .	67	Processing Routines . . . . .	100
Moving Information into the		Non-Debugging Declarative	
Dictionary . . . . .	67	Considerations . . . . .	102
Priority Checking for Segmentation . . . . .	69	Output . . . . .	103
Verbs . . . . .	69	PHASE 4 . . . . .	104
Procedure-Branching Verbs . . . . .	69	Translation of P1-Text to P2-Text	104
Input/Output Verbs . . . . .	69	Verbs . . . . .	104
Other Verbs . . . . .	70	MOVE Statement -- Subscripting . . . . .	104
Declaratives . . . . .	70	DEBUG Card . . . . .	105
		ALTER Statement . . . . .	105
PHASE 20 . . . . .	71	PERFORM Statement . . . . .	109
Translating LD Entries into ATF-text . . . . .	72	COMPUTE Statement . . . . .	111
Processing Elementary Items . . . . .	72	Multiple Results in Arithmetic	
Processing Group Items . . . . .	72	Statements . . . . .	113
Producing Incomplete Data A-text . . . . .	72	IF Statement . . . . .	113
Processing File Section Entries . . . . .	72	Nested IF Statement . . . . .	113
Processing Communication Section		SEARCH Format-2 (SEARCH ALL)	
Entries . . . . .	73	Statement . . . . .	113
Processing Errors . . . . .	73	Syntax Analysis and Error Checking . . . . .	116
		Method of Defining Verb Blocks . . . . .	116
PHASE 22 . . . . .	74	Phase 40 Initialization Routine . . . . .	116
Building Dictionary Entries . . . . .	74	ID5 Routine . . . . .	116
Dictionary Preprocessing . . . . .	74	ISTRUV Routine . . . . .	116
Completing Dictionary Entries . . . . .	76	IDBRK Routine . . . . .	116
Generating Data A-Text . . . . .	77	IDLH03 Routine . . . . .	116
Q-Routine Generation . . . . .	77	SORT, MERGE Routines . . . . .	116
Processing Errors . . . . .	78	EOF Routine . . . . .	116
Building Tables for Later Phases . . . . .	78	GETNXT (GET13 and GET14) Routine . . . . .	117
		EXIT5 (EXIT PROGRAM) Routine . . . . .	117
PHASE 21 . . . . .	79	GENNOD Routine . . . . .	117
FD Dictionary Entries . . . . .	79	GENPAR Routine . . . . .	117
SD Dictionary Entries . . . . .	80	GENTIM Routine . . . . .	117
Linage-Counter Entries . . . . .	80	WRSYS4 Routine . . . . .	117
Data A-text Elements . . . . .	80		
FIB Address Element . . . . .	80	PHASE 45 . . . . .	118
Block and Working-Storage Section		Initialization and ATM-Text Analysis . . . . .	118
Address Elements . . . . .	80	Creating P2-Text for Phases 50 and 51 . . . . .	118
Constant and Address Constant			
Definition Elements . . . . .	80	PHASE 50 . . . . .	119
Creation of Exit Lists . . . . .	80	Program Breaks . . . . .	119
DCB's and DECB's -- Address and		Verb Strings . . . . .	119
Constant Definition Elements . . . . .	80	Verb Processing . . . . .	120
File Information Block (FIB) . . . . .	80	Resolving Subscripted and Indexed	
Determination of Buffer Area Size . . . . .	81	References . . . . .	120
Clause Compatibility . . . . .	84	How Subscripted Addresses Are	
		Calculated . . . . .	121
PHASE 25 . . . . .	85	Using and Optimizing Subscript	
Phase 25 Processing for the Debug		References . . . . .	122
Data Set . . . . .	85	Indexed References . . . . .	123
Building the OBODOTAB Table . . . . .	85	Arithmetic Verb Strings . . . . .	124
Building the DATATAB Table . . . . .	86	Work Area . . . . .	124
		Compile-Time Arithmetic . . . . .	125
PHASE 3 . . . . .	88	Mode of Operation . . . . .	126
Glossary Building . . . . .	88	Register and Storage Allocation . . . . .	126
Translation from P0-text to P1-text . . . . .	89	Generating A-text . . . . .	127
READ Verb Strings . . . . .	89		



Literals and Virtuals . . . . .	131	Building the VN Priority Table . . . . .	165
Handling Phase 51 Verb Strings . . . . .	132	Processing PNs and GNs . . . . .	165
Additional Processing for the OPT Option . . . . .	132	Optimizing Literals and Processing DISPLAY Literals . . . . .	165
PHASE 51 . . . . .	133	Optimizing Virtuals . . . . .	166
E-Text . . . . .	133	Allocating Storage for the Program Global Table . . . . .	166
Program Breaks . . . . .	134	DEBUG LINKAGE AREA Allocation . . . . .	166
Segmentation Control Breaks . . . . .	134	OVERFLOW Allocation . . . . .	166
PN, GN, and VN Definitions . . . . .	134	VIRTUAL Allocation . . . . .	167
Building PN and GN Equate Strings . . . . .	134	VIRTUAL EBCDIC NAMES Allocation . . . . .	167
Building the PNOTBL Table . . . . .	135	PN Allocation . . . . .	167
Verb Strings . . . . .	135	GN Allocation . . . . .	167
Input/Output Verbs . . . . .	135	DCBADR Allocation . . . . .	168
Other Nonarithmetic Verb Strings . . . . .	137	VNI Allocation . . . . .	168
Special Considerations for Nonarithmetic Verbs . . . . .	138	LITERAL Allocation . . . . .	168
Verbs Requiring Calls to Object-Time Subroutines . . . . .	139	PROCEDURE BLOCK Allocation . . . . .	168
DISPLAY Literals . . . . .	139	Optimizing Register Assignments . . . . .	168
Generating System/370 Instructions . . . . .	140	Permanent Register Assignments . . . . .	169
Text Generation for the OPT Option . . . . .	140	Temporary Register Assignments . . . . .	169
Generating Object Code to Process VSAM Files . . . . .	140	Optimizing and Allocating Storage for the Procedure Division . . . . .	171
Generating Calls to the ILBOVCOO and ILBOVIOO Subroutines . . . . .	140	Building the PNLABTBL and GNLABTBL Tables . . . . .	171
PHASE 6 . . . . .	141	Incrementing the ACCUMCTR Counter . . . . .	171
Output of Phase 6 . . . . .	141	PHASE 63 . . . . .	175
Suppression of Output Listing . . . . .	142	Initialization of Phase 63 . . . . .	175
Glossary and Listing Symbols . . . . .	142	Constructing Procedure A1-text . . . . .	175
Task Global Table Storage Allocation . . . . .	143	Control Routine . . . . .	175
Optimizing Storage for the Program Global Table . . . . .	144	Processing Programs with One Procedure Block . . . . .	175
Building the VN Priority Table . . . . .	144	Processing for Branch Instructions . . . . .	175
Optimizing PNs and GNs . . . . .	146	Processing for Optimization Information Elements (C001-C007) . . . . .	176
Optimizing Literals and DISPLAY Literals . . . . .	147	Processing for RPT-Origin (D4) Element . . . . .	176
Optimizing Virtuals . . . . .	147	Processing for Address Reference (78) Elements . . . . .	176
Allocating Storage for the Program Global Table . . . . .	148	Processing for Address Increment (80) Elements . . . . .	176
OVERFLOW Allocation . . . . .	149	Processing for Incremented Address (A4) Elements . . . . .	176
VIRTUAL Allocation . . . . .	149	Constructing Debug-text . . . . .	176
VIRTUAL EBCDIC NAMES Allocation . . . . .	149	Counters Used in Phase 63 . . . . .	177
PN Allocation . . . . .	149	Building the QGNTBL Table . . . . .	177
GN Allocation . . . . .	150	Making Entries in the RLDTBL Table . . . . .	177
DCBADR Allocation . . . . .	150	Processing in a Segmented Program . . . . .	177
VNI Allocation . . . . .	150	Processing at End of File . . . . .	178
LITERAL Allocation . . . . .	150	PHASE 64 . . . . .	179
Procedure A-text Processing . . . . .	151	Output of Phase 64 . . . . .	179
Procedure A-text Processing in a Segmented Program . . . . .	155	Processing Data A-text, E-text, and DEF-text . . . . .	179
Listing A-Text . . . . .	156	Processing Procedure A1-text . . . . .	179
Execution-Time Base Register Assignment . . . . .	156	Initialization Coding Generation . . . . .	183
Processing Data A-text, E-text, and DEF-text . . . . .	157	Processing the RLDTBL Table . . . . .	183
Processing the RLDTBL Table . . . . .	157	PHASE 65 . . . . .	187
Initialization Coding Generation . . . . .	157	Processing the FLOW Option . . . . .	187
PHASE 62 . . . . .	161	Common Processing For The STATE, SYMDMP, And TEST Options . . . . .	187
Output of Phases 62, 63, and 64 . . . . .	161	Processing Debug-text . . . . .	187
Suppression of Output Listing . . . . .	162	Building the PROCTAB Table . . . . .	187
Task Global Table Storage Allocation . . . . .	162	Building the SEGINDX Table . . . . .	188
Optimizing Storage for the Program Global Table . . . . .	164	Further Processing for the STATE Option . . . . .	188
		Further Processing For The SYMDMP Or TEST Options . . . . .	188

Building the CARDINDX Table . . . . .	188	SD Entry . . . . .	449
Building the PROCINDX Table . . . . .	188	RD Entry . . . . .	449
Debug Data Set Processing . . . . .	188	LD Entry . . . . .	450
Final Processing . . . . .	189	CD Entry . . . . .	450
PHASE 6A . . . . .	190	Condition-Name Entry . . . . .	454
Producing a Source Ordered		Index-Name Entry . . . . .	454
Cross-Reference Listing . . . . .	190	DEBUG DATA SET TABLES . . . . .	455
Producing an Alphabetically Ordered		PROGSUM Table . . . . .	456
Cross-Reference Listing . . . . .	191	OBODOTAB Table . . . . .	457
PHASES 70, 71, AND 72 . . . . .	192	DATATAB Table . . . . .	458
Input from Prior Phases . . . . .	192	PROCTAB Table . . . . .	465
Phase 70 Error Processing . . . . .	192	CARDINDX Table . . . . .	465
PARTBL and EACTBL Tables . . . . .	193	SEGINDX Table . . . . .	466
PHXERR Tables . . . . .	193	PROCINDX Table . . . . .	466
Generating Messages . . . . .	193	BCDPN Table . . . . .	466
Error Message Listing . . . . .	194	VSAM File Information Block (FIB) . . . . .	467
FIPS Processing . . . . .	194	SECTION 6. DIAGNOSTIC AIDS . . . . .	470
PHASE 80 . . . . .	195	Procedure To Force Core Dump . . . . .	470
Input . . . . .	195	Response to System Error Recovery	
Output . . . . .	195	Findings . . . . .	470
Processing . . . . .	195	Compiler Data Set Activity . . . . .	470
Scanning the Source Program . . . . .	195	Register Usage by Each Phase . . . . .	470
Generating Diagnostic Messages . . . . .	195	Register Assignment . . . . .	483
Writing the Source Program . . . . .	196	Elements of Program Design . . . . .	483
SECTION 3. PROGRAM ORGANIZATION . . . . .	197	Compiler Error Messages . . . . .	483
Flowcharts . . . . .	197	ABEND Codes . . . . .	483
SECTION 4. DIRECTORY . . . . .	309	Identifying the Version of the	
Flowchart Label Directory . . . . .	309	Compiler . . . . .	486
Tables Used by Phases . . . . .	313	Current Phase and Record . . . . .	486
MICROFICHE DIRECTORIES . . . . .	317	Saving Registers . . . . .	487
SECTION 5. DATA AREAS . . . . .	324	Buffers and Their Contents . . . . .	487
Communications Area (COMMON) . . . . .	324	Locating Tables . . . . .	488
COMPILER TABLE FORMATS . . . . .	340	Diagnostic Assistance . . . . .	488
Notes on Compiler Table Formats . . . . .	340	APPENDIX A: TABLE AND DICTIONARY	
INTERNAL TEXT FORMATS . . . . .	399	HANDLING (TAMER) . . . . .	492
Notes on Text Element Formats . . . . .	399	Access Dictionary Handling Routines . . . . .	492
IPTEXT . . . . .	400	Organization of the Dictionary . . . . .	492
Data IC-Text . . . . .	402	Storage for the Dictionary . . . . .	493
ATF-Text . . . . .	408	Initialization of ACCESS Routines . . . . .	493
Data A-Text . . . . .	409	ACCESS Routines . . . . .	493
Procedure IC-Text (P0 Format) . . . . .	411	ENTNAM (Enter Attributes Given	
Procedure IC-Text (P1 Format) . . . . .	419	Name) . . . . .	494
PROCEDURE P1A-TEXT . . . . .	424	ENTPTR (Enter Attributes Given	
Procedure IC-Text (P2 Format) . . . . .	425	Pointer) . . . . .	494
ATM-Text . . . . .	433	GETPTR (Get Pointer) . . . . .	494
Procedure A-text . . . . .	434	ENTDEL (Enter Delimiter Pointer) . . . . .	494
Optimization A-Text . . . . .	439	LATRNM (Locate Attributes Given	
Procedure A1-Text . . . . .	441	Name) . . . . .	495
Listing A-Text . . . . .	442	LATRPT (Locate Attributes Given	
E-Text . . . . .	442	Pointer) . . . . .	495
XREF-Text . . . . .	443	LOCNXT (Locate Next Entry) . . . . .	495
Debug-Text . . . . .	444	LDELNM (Locate Delimiter Given	
DICTIONARY ENTRY FORMATS . . . . .	445	Name) . . . . .	495
Notes on Dictionary Entry Formats . . . . .	445	LATACP (Locate Attributes Using	
Procedure-Name (Paragraph) Entry . . . . .	446	ACCESS Pointer) . . . . .	496
Procedure-Name (Section) Entry . . . . .	446	LATGRP (Locate Attributes Given	
FD Entry . . . . .	447	Group Pointer) . . . . .	496
		Table Handling with TAMER . . . . .	496
		CONTROL Fields . . . . .	497
		TIBS (Table Information Blocks) . . . . .	497
		TAMMS (Table Area Management Maps) . . . . .	497
		MASTAMS (Master TAMM Tables) . . . . .	497
		How Space is Assigned . . . . .	498
		TAMEIN Routine . . . . .	498
		PRIME Routine . . . . .	498

TBGETSPC Routine . . . . .	499	CFF-ROUT Routine . . . . .	521
MOVDIC Routine . . . . .	499	PGH-ROUT Routine . . . . .	521
DICSPC Routine . . . . .	499	PGF-ROUT Routine . . . . .	521
STATIC Routine . . . . .	499	DET-ROUT Routine . . . . .	522
TABREL Routine . . . . .	499	Data-Names . . . . .	522
INSERT Routine . . . . .	500	COBOL Word Data-names . . . . .	522
TAMEOP Routine . . . . .	500	Nonstandard Data-names . . . . .	522
TBSPILL Routine . . . . .	500	Special Report Writer Verbs . . . . .	523
TBWRITE Routine . . . . .	500	Response to Procedure Division Verbs . . . . .	524
TBREADIC Routine . . . . .	500	Finding the Elements of a Report Writer	
GETALL Routine . . . . .	500	Subprogram (RWS) . . . . .	524
		Locating Data Items in a Storage Dump	524
APPENDIX B: OBJECT MODULE . . . . .	501	Locating Data Items in the Object	
Overview of Object Module Fields . . . . .	501	Module . . . . .	524
Initialization 1 Routine (INIT1) . . . . .	501	Locating Routines in a Storage Dump	524
Data Area . . . . .	502	Locating Routines in the Object	
Exit Lists . . . . .	502	Module . . . . .	527
Task Global Table (TGT) . . . . .	505	Locating DET-ROUT and USM-ROUT	
Program Global Table (PGT) . . . . .	510	Routines . . . . .	527
Report Writer Routines (RPT) . . . . .	512	Locating CTF-ROUT and CTH-ROUT	
Procedure . . . . .	512	Routines . . . . .	528
Q-Routines . . . . .	512		
COUNT Table . . . . .	512	APPENDIX D: INTERFACE WITH	
Initialization 2 Routine (INIT2) . . . . .	513	CONVERSATIONAL MONITOR SYSTEM (CMS) . . . . .	529
Initialization 3 Routine (INIT3) . . . . .	513	Introduction . . . . .	529
PROCTAB Table (PROCTAB) . . . . .	514	Functions . . . . .	529
SEGINDX Table (SEGINDX) . . . . .	514	Environment . . . . .	529
Segmented Object Module (TRANSIENT		Physical Characteristics . . . . .	529
AREA) . . . . .	514	Operational Considerations . . . . .	531
APPENDIX C: REPORT WRITER SUBPROGRAM . . . . .	515	Source Program Filename . . . . .	531
Structure of the Report Writer		Option List . . . . .	531
Subprogram (RWS) . . . . .	515	Issuing CMS FILEDEF Commands . . . . .	532
Elements of a Report Writer Subprogram		Method of Operation . . . . .	532
(RWS) . . . . .	515	Initialization . . . . .	533
Fixed Routines . . . . .	515	Special Processing for TFXE and	
1ST-ROUT Routine . . . . .	515	SYSPUNCH Files . . . . .	533
LST-ROUT Routine . . . . .	515	Compiler Directory Information . . . . .	533
WRT-ROUT Routine . . . . .	515	Error Processing . . . . .	533
Parametric Routines . . . . .	520	Returning Control to the CMS Command	
USM-ROUT Routine . . . . .	520	Environment . . . . .	535
CTB-ROUT Routine . . . . .	520	Program Organization . . . . .	535
ROL-ROUT Routine . . . . .	520	Directories . . . . .	535
RST-ROUT Routine . . . . .	520	Flowchart Label Directory . . . . .	539
SAV-ROUT Routine . . . . .	520	Diagnostic Aids . . . . .	539
RET-ROUT Routine . . . . .	520	Data Set Activity . . . . .	539
INT-ROUT Routine . . . . .	520	Register Usage . . . . .	539
ALS-ROUT Routine . . . . .	520	Elements of Program Design . . . . .	539
RLS-ROUT Routine . . . . .	521	Error Messages Issued by DMSCOB . . . . .	539
Group Routines . . . . .	521	CMS Service Routines Called by	
RPH-ROUT Routine . . . . .	521	DMSCOB . . . . .	541
RPF-ROUT Routine . . . . .	521	Register Saving . . . . .	541
CTH-ROUT Routine . . . . .	521		
CTF-ROUT Routine . . . . .	521	GLOSSARY . . . . .	542
CHF-ROUT Routine . . . . .	521	DIAGRAMS . . . . .	547
		INDEX . . . . .	587



FIGURES

Figure 1. Flow of Control at End of Compilation . . . . .	32	Figure 32. Parameter Cells for the A-Text Generator (Part 1 of 4) . . . . .	128
Figure 2 (Part 1 of 2). Linkage Codes to Phase 00 . . . . .	33	Figure 33. Analysis of an ON Statement . . . . .	138
Figure 3. Flow of Control for Processing Between Phases . . . . .	35	Figure 34. Analysis of a DISPLAY Verb Listing and Glossary to Define Compiler-Generated Information . . . . .	139
Figure 4. Optional Phase Processing . . . . .	36	Figure 35. Symbols Used in the Compiler-Generated Information . . . . .	143
Figure 5. Activity of the Compiler Data Sets and Buffer Assignments (Part 1 of 7) . . . . .	37	Figure 36. Use of Counters in COMMON to Allocate Space in the TGT for Variable-length Fields (Part 1 of 2) . . . . .	145
Figure 6. Table Usage During Record Description Processing . . . . .	60	Figure 37. PNTBL, PNTBL, and GNTBL Tables at the Beginning of Optimization Processing . . . . .	146
Figure 7. Phase 12 Input/Output Flow . . . . .	63	Figure 38. GNTBL Table after PN and GN Equate Strings Have Been Processed . . . . .	147
Figure 8. Entering PNTABL and PNQTLB Information in the Dictionary . . . . .	68	Figure 39. GNTBL Table after the Relative Numbers Have Been Assigned . . . . .	147
Figure 9. Phase 20 Input/Output Flow . . . . .	71	Figure 40. CONTBL, CONDIS, and LTLTBL Tables after Processing Literals . . . . .	148
Figure 10. Phase 22 Input/Output Flow . . . . .	75	Figure 41. CVIRTB and VIRPTR Tables after Processing Virtuals . . . . .	148
Figure 11. Building the OBODOTAB Table . . . . .	87	Figure 42. VIRPTR Table after VIRTUAL Allocation . . . . .	149
Figure 12. P1-text Resulting from an ADD CORRESPONDING Option . . . . .	90	Figure 43. PNTBL and GNTBL Values after PGT Allocation . . . . .	150
Figure 13. P1-text Resulting from a MOVE CORRESPONDING Option . . . . .	90	Figure 44. LTLTBL Table after Literal Allocation . . . . .	151
Figure 14. P1-text Written for SEARCH Format-1 P0-text (Part 1 of 2) . . . . .	92	Figure 45. Processing Procedure A-text Elements (Part 1 of 3) . . . . .	152
Figure 15. P1-text Written for SEARCH Format-2 P0-text . . . . .	94	Figure 46. Contents of SYSUT4 When Read by Phase 6 . . . . .	158
Figure 16. P1-text Written for Condition-String Testing Multiple Values without Using the VALUE...THRU Clause . . . . .	97	Figure 47. Processing Data A-text, E-text, and DEF-text (Part 1 of 2) . . . . .	159
Figure 17. P1-text Written for Condition-String with VALUE...THRU Clause . . . . .	98	Figure 48. Use of Counters in COMMON to Allocate Space in the TGT for Variable-length Fields (Part 1 of 2) . . . . .	163
Figure 18. Tables and Output for the Statement MOVE A(6) TO B(C,D,E) . . . . .	105	Figure 49. CONTBL, CONDIS, and LTLTBL Tables after Processing Literals . . . . .	165
Figure 19. DBGTLB Entries and P2-text for DEBUG Card Processing . . . . .	105	Figure 50. CVIRTB and VIRPTR Tables after Processing Virtuals . . . . .	166
Figure 20. Table Entries and Output for ALTER Statements . . . . .	107	Figure 51. VIRPTR Table after VIRTUAL Allocation . . . . .	167
Figure 21. Execution of an ALTER Statement . . . . .	107	Figure 52. LTLTBL Table after Literal Allocation . . . . .	168
Figure 22. Flow of Control for ALTER/GO TO Statements . . . . .	108	Figure 53. Optimizing Assignment of Registers 14 and 15 . . . . .	170
Figure 23. Effect of a PERFORM Statement . . . . .	109	Figure 54. Processing for Optimization Information Elements (Part 1 of 3) . . . . .	172
Figure 24. Execution of a PERFORM Statement . . . . .	110	Figure 55. Contents of SYSUT4 when read by Phase 64 . . . . .	180
Figure 25. Flow of Control for a PERFORM Statement . . . . .	111	Figure 56. Processing Data A-text, E-text, and DEF-text (Part 1 of 2) . . . . .	181
Figure 26. Evaluation of a COMPUTE Statement . . . . .	112	Figure 57. Processing Procedure A1-text Elements (Part 1 of 4) . . . . .	183
Figure 27. Strings Resulting from a COMPUTE Statement . . . . .	112	Figure 58. Explanation of Flowchart Symbols . . . . .	198
Figure 28. Evaluation of a Nested IF Statement . . . . .	113	Figure 59. Tables Used by Phases (Part 1 of 2) . . . . .	313
Figure 29. Example of Phase 4 Output for a SEARCH ALL Statement . . . . .	114	Figure 60. TIB Usage . . . . .	315
Figure 30. Flow of Execution for a SEARCH ALL Statement . . . . .	115		
Figure 31. Arithmetic Processing Switches . . . . .	125		

Figure 61. Types of compiler Text Produced by Each Phase . . . . .	316	Figure 78. Fields of the Program Global Table . . . . .	510
Figure 62. Load Module Directory (Part 1 of 4) . . . . .	317	Figure 79 (Part 1 of 4). Logic of the Generated Report Writer Subprogram . .	516
Figure 63 (Part 1 of 5). External Symbol Directory . . . . .	321	Figure 80. First GENERATE Statement Logic Flow . . . . .	525
Figure 64. Registers Pointing to COMMON . . . . .	324	Figure 81. Logic Flow of All GENERATE Statements After the First . . . . .	526
Figure 65 (Part 1 of 2). LD Entry Variable Information . . . . .	452	Figure 82. TERMINATE Statement Logic Flow . . . . .	527
Figure 66. SYSUT5 (Debug Data Set) . .	455	Figure 83. Report Writer Subprogram GN Numbers . . . . .	528
Figure 67 (Part 1 of 12). Register Usage According to Phase . . . . .	471	Figure 84. Relationships Among CMS-COBOL Interface Routine, the COBOL Compiler, and CMS . . . . .	530
Figure 68. Register Usage at Execution	483	Figure 85. COBOL Compiler Options Under CMS . . . . .	531
Figure 69. Register Usage at Execution (OPT) . . . . .	483	Figure 86. FILEDEF Commands Issued for Compilation Under CMS . . . . .	532
Figure 70 (Part 1 of 2). Error Messages Indicating Compiler Error . .	485	Figure 87. Operations of DMSCOB Routine at Initialization . . . . .	534
Figure 71. Location of Identifier Constant . . . . .	486	Figure 88. Load Module Directory . .	535
Figure 72. POINT Table Entry Format .	488	Figure 89. External Symbol Directory	535
Figure 73. Arrangement of Tables and Dictionary Sections in Contiguous Areas	492	Figure 90 (Part 1 of 2). Flowchart Label Directory . . . . .	539
Figure 74. Storage Map of a COBOL Object Module . . . . .	501	Figure 91. Register Usage by DMSCOB .	540
Figure 75. Format of the Data Area . .	502	Figure 92. Error Messages Issued by DMSCOB . . . . .	540
Figure 76. Fields of the Exit List . .	503	Figure 93. CMS Service Routines Called by DMSCOB . . . . .	541
Figure 77. Fields of the Task Global Table . . . . .	505		

List of Charts

Chart AA (Part 1 of 7). Phase 00:	Chart DK. Phase 22: READP4 Routine	.243
Overall Flow	Chart DL. Phase 22: DICTBD Routine	.244
Chart AA (Part 2 of 7). Phase 00:	Chart DM. Phase 21: Overall Flow	.245
READ Routine	Chart DN. Phase 21: FILEST Routine	.246
Chart AA (Part 3 of 7). Phase 00:	Chart DO. Phase 25: Overall Flow	.247
WRITEA, WRITE, and WOUT Routines	Chart DP. Phase 25: ODOBLD,	
Chart AA (Part 4 of 7). Phase 00:	BLDOBODO, and ENDP1 Routines	.248
WRITEA, WRITE, and WOUT Routines	Chart DQ. Phase 25: BEGPASS Routine	.249
Chart AA (Part 5 of 7). Phase 00:	Chart DR. Phase 25: TESTSUBS and	
WPCB and WGO Routines	SETNAMS Routines	.250
Chart AA (Part 6 of 7). Phase 00:	Chart EA. Phase 3: Overall Flow	.251
READ Library Routines	Chart EB. Phase 3: GLOSRY Routine	.252
Chart AA (Part 7 of 7). Phase 00:	Chart EC. Phase 3: PHCTRL Routine	.253
PLSCALL Routines	Chart ED (Part 1 of 5). Phase 35:	
Chart BA. Phase 01: Overall Flow	PHCTRL Main Control Routine	.254
Chart BB. Phase 02: Overall	Chart ED (Part 2 of 5). Phase 35:	
Chart BC. Phase 03: Overall Flow	ANLZUFDS Routine	.255
Chart BD (Part 1 of 4). Phase 04:	Chart ED (Part 3 of 5). Phase 35:	
IKFCBL04	PNDEFRTN Routine	.256
Chart BD (Part 2 of 4). Phase 04:	Chart ED (Part 4 of 5). Phase 35:	
BASISRTN	GOTAVERB Routine	.257
Chart BD (Part 3 of 4). Phase 04:	Chart ED (Part 5 of 5). Phase 35:	
COPYRTN	ANLZVRBS Routine	.258
Chart BD (Part 4 of 4). Phase 04:	Chart FA. Phase 4: Overall Flow	.259
COPYPROC	Chart FB. Phase 4: IF Routine	.260
Chart BE (Part 1 of 3). Phase 05:	Chart FC. Phase 4: PRFORM Routine	.261
Overall Flow	Chart FD. Phase 45: Overall Flow	.262
Chart BE (Part 2 of 3). Phase 05:	Chart FE. Phase 45: UNSTRING Routine	.263
Language Analysis Routine	Chart FF. Phase 45: SORTXT Routine	.264
Chart BE (Part 3 of 3). Phase 05:	Chart GA. Phase 50: Overall Flow	.265
Input and Scanning Routines (SCAN)	Chart GB. Phase 50: PH5CTL Routine	.266
Chart BF (Part 1 of 4). Phase 06:	Chart GC (Part 1 of 2). Phase 50:	
Overall Flow	GETNXT Routine	.267
Chart BF (Part 2 of 4). Phase 06:	Chart GC (Part 2 of 2). Phase 50:	
IPTEXT ITEM Processors	GETNXT Routine	.268
Chart BF (Part 3 of 4). Phase 06:	Chart GD. Phase 50: Generate Routine	.269
IPTEXT ITEM Processors	Chart GE. Phase 50: XSPRO and KILSUB	
Chart BF (Part 4 of 4). Phase 06:	Routines	.270
IPTEXT ITEM Processors	Chart GF. Phase 51: Overall Flow	.271
Chart BG (Part 1 of 2). Phase 08:	Chart GG. Phase 51: DBGTEST Routine	.272
Overall Flow	Chart GH. Phase 51: GETNXT Routine	.273
Chart BG (Part 2 of 2). Phase 08:	Chart GI. Phase 51: PUTDEF Routine	.274
Data Division Flow	Chart GJ. Phase 51: A-text Generator	
Chart CA. Phase 10: Overall Flow	Routine	.275
Chart CB. Phase 10: IDDCSN Routine	Chart HA. Phase 6: Overall Flow	.276
Chart CC. Phase 10: ENVSCN Routine	Chart HB. Phase 6: PH6 Routine	.277
Chart CD. Phase 10: DDSCN Routine	Chart HC. Phase 6: PRFTWO Routine	.278
Chart CE. Phase 10: Overall Flow	Chart HD. Phase 6: SE6000 Routine	.279
Chart CF. Phase 12: RDSCAN Routine	Chart HE. Phase 6: PDATEX Routine	.280
Chart CG. Phase 12: PROC01 Routine	Chart HF. Phase 6: GINIT2 Routine	.281
Chart CH. Phase 12: PROC02 Routine	Chart IA. Phase 62: Overall Flow	.282
Chart CI. Phase 12: FLUSH Routine	Chart IB. Phase 62: PH6 Routine	.283
Chart CJ. Phase 12: GNSPRT and	Chart IC. Phase 62: PRFTWO Routine	.284
SPCRTS Routines	Chart ID (Part 1 of 2). Phase 62:	
Chart CK. Phase 1B: Overall Flow	SE6000 Routine	.285
(PDSCN Routine)	Chart ID (Part 2 of 2). Phase 62:	
Chart DA. Phase 20: Overall Flow	SE6000 Routine	.286
Chart DB. Phase 20: FILEST Routine	Chart IE. Phase 63: Overall Flow	.287
Chart DC. Phase 20: WSTSCT, LINKST,	Chart IF. Phase 63: BRANCH Routine	.288
COMSCT, and REPORT Routines	Chart IG. Phase 63: GNDEF Routine	.289
Chart DD. Phase 20: LDTEXT Routine	Chart IH. Phase 63: PNDEF Routine	.290
Chart DE. Phase 22: Overall Flow	Chart II. Phase 63: ADREF Routine	.291
Chart DF. Phase 22: FSECT Routine	Chart IJ. Phase 63: C1REF Routine	.292
Chart DG. Phase 22: WSECT and LSECT	Chart IK. Phase 64: Overall Flow	.293
Routines	Chart IL. Phase 64: PDATEX Routine	.294
Chart DH. Phase 22: CDSECT Routine	Chart IM. Phase 64: SE6000 Routine	.295
Chart DI. Phase 22: RSECT Routine	Chart IN. Phase 64: ADREF, RC4,	
Chart DJ. Phase 22: LDXTX Routine	RC8C, and RD001 Routines	.296

Chart IO. Phase 64: GINIT2 Routine . . . . .	.297	Chart KA (Part 2 of 5). Phase 80: FIPS . . . . .	.305
Chart IP. Phase 65: Overall Flow . . . . .	.298	Chart KA (Part 3 of 5). Phase 80: FIPS . . . . .	.306
Chart IQ. Phase 65: TENPROC, TWENPROC, and GTEQ10K Routines . . . . .	.299	Chart KA (Part 4 of 5). Phase 80: FIPS . . . . .	.307
Chart IR (Part 1 of 3). Phase 6A: Overall Flow . . . . .	.300	Chart KA (Part 5 of 5). Phase 80: FIPS . . . . .	.308
Chart IR (Part 2 of 3). Phase 6A: Overall Flow . . . . .	.301	Chart VM (Part 1 of 3). DMSCOB (COBOL-CMS Interface Routine) . . . . .	.536
Chart IR (Part 3 of 3). Phase 6A: Overall Flow . . . . .	.302	Chart VM (Part 2 of 3). DMSCOB (COBOL-CMS Interface Routine) . . . . .	.537
Chart JA. Phases 70, 71, and 72: Overall Flow . . . . .	.303	Chart VM (Part 3 of 3). DMSCOB (COBOL-CMS Interface Routine) . . . . .	.537
Chart KA (Part 1 of 5). Phase 80: FIPS . . . . .	.304		



Operation Diagrams

Diagram 1. Part 1. Design of the OS/VS COBOL Compiler . . . . .	549	Diagram 2. Part 5. Method of Operation: Procedure Division Translation . . . . .	567
Diagram 1. Part 2. Design of the OS/VS COBOL Compiler . . . . .	551	Diagram 2. Part 6. Method of Operation: Object Module Production . . . . .	569
Diagram 1. Part 3. Design of the OS/VS COBOL Compiler . . . . .	553	Diagram 2. Part 7. Method of Operation: Optimization of Object Module (Optional) . . . . .	571
Diagram 2. Part 1. Method of Operation: Table of Contents . . . . .	555	Diagram 2. Part 8. Method of Operation: Debug Data Set Creation (Optional) . . . . .	573
Diagram 2. Part 2. Method of Operation: Overview . . . . .	557	Diagram 2. Part 9. Method of Operation: Error Messages, Diagnostics, and Cross-Reference Listings . . . . .	575
Diagram 2. Part 3. Method of Operation: Control and Input/Output . . . . .	559	Diagram 3. Phase 25 Operations . . . . .	577
Diagram 2. Part 3a. COPY and BASIS Processor . . . . .	561	Diagram 4. Phase 3 Operations . . . . .	579
Diagram 2. Part 3B. Reformatted Source Code Listing and Embedded Cross-references . . . . .	563	Diagram 5. Phase 62 Operations . . . . .	581
Diagram 2. Part 4. Method of Operation: Identification, Environment, and Data Division Translation . . . . .	565	Diagram 6. Phase 63 Operations . . . . .	583
		Diagram 7. Phase 65 Operations . . . . .	585



SECTION 1. INTRODUCTION

The IBM OS/VS COBOL Compiler analyzes source modules written in the COBOL language and translates them into object programs suitable for input to the linkage editor for subsequent execution on the computer. This publication describes the design of this compiler and the characteristics of the object program which it produces.

RELATIONSHIP OF THE COMPILER TO THE IBM OS/VS SYSTEM

A COBOL compilation is a job step under the control of the operating system. The OS/VS COBOL compiler can also be invoked under the Time Sharing Option (TSO) of the IBM Operating System. The compiler, operating under TSO, can be invoked most conveniently by the Program Product, TSO COBOL Prompter, Program Number 5734-CP1. The OS/VS COBOL compiler can also be invoked under the IBM Virtual Machine Facility/370 Conversational Monitor System (CMS).<sup>1</sup> To use the compiler, see the publication IBM OS/VS COBOL Compiler and Library Programmer's Guide, which explains how a COBOL compilation is introduced to the operating system.

As a processing program of the IBM Operating System, the compiler communicates with the control program of the operating system for input/output and other services. The services provided by the control program are described in the publications IBM OS/VS Supervisor Services and Macro Instructions, IBM OS/VS Data Management Services Guide, IBM OS/VS Data Management Macro Instructions.

CHARACTERISTICS OF THE COMPILER

PHYSICAL CHARACTERISTICS

The compiler consists of 31 phases; from 10 to 14 of these phases perform the actual transformation of a source module into an object program. Phase 04 is called only if

<sup>1</sup>The COBOL-CMS Interface routine, which allows compilation under CMS, is described in "Appendix D: Interface with Conversational Monitor System (CMS)."

there is COPY or BASIS processing to be performed. Phases 05, 06, and 08 are called only if the Lister (LSTCOMP or LSTONLY) option is in effect. Phase 12 is called only if a Report Section appears in the Data Division. Phase 35 is called only if WITH DEBUGGING MODE and USE FOR DEBUGGING declaratives are present. Phase 45 is entered only if an UNSTRING verb is encountered in the source program. If optimization of the object code has been requested through the OPT option, phases 62, 63, and 64 replace phase 6. Phase 80 is called to flag source statements which do not meet the Federal Information Processing Standard when the LVL option is specified.

Phases 25, 65, 6A, 70, 71, and 72 are also optional phases: phases 25 and 65 generate debugging information for the SYMDMP, FLOW, and/or STATE options; phase 6A produces a cross-reference listing if the user requests one; and phases 70, 71, and 72 are used to list the error messages if errors were found in the source module.

Of the other phases, phase 00 acts as the interface between the compiler and the operating system, phase 01 contains the installation default values of compilation parameters, phase 02 performs compiler initialization, and phase 03 issues error messages for terminal error conditions.

The phases are organized into an overlay structure, but the overlays are controlled through phase 00. Phase 00 is resident throughout compilation. It links to other phases as they are needed. The linkage sequence is as follows:

Phase 00 links to phases

- 01
- 03 (if disaster condition with NODUMP)
- 04 (if COPY or BASIS)
- 05 (if LSTCOMP or LSTONLY)
- 06 (if LSTCOMP or LSTONLY)
- 08 (if LSTCOMP or LSTONLY)
- 10
- 12 (if report writer)
- 1B
- 20
- 22
- 21
- 25 (if SYMDMP TEST)
- 3
- 35 (if USE FOR DEBUGGING)
- 4
- 45 (if UNSTRING)

- 50
- 51
- 6 (if NOOPT)
- 62 (if OPT)
- 63 (if OPT)
- 64 (if OPT)
- 65 (if FLOW, STATE, SYM, or TEST)
- 6A (if SXREF, VBREF, VBSUM, or XREF)
- 70 (if any error message from phases 04 through 6A)
- 80 (if LVL)

Phase 01 links to phase 02.  
Phase 70 links to phases 71 and 72.

## OPERATIONAL CHARACTERISTICS

### Input

Input to the compiler consists of an OS/VIS COBOL program, written in the language described in IBM VS COBOL for OS/VIS. The source program is read from the SYSIN and, optionally, the SYSLIB or other library data sets. Other input consists of control cards for BASIS and COPY and for batch multiple compilations.

### Output

The output of the compiler depends upon the options specified by the user on the EXEC statement, in the COBOL command string, or at installation time. It may consist of:

1. A source program listing on SYSPRINT<sup>1</sup>
2. Compilation progress messages and error messages on SYSTEM and error messages on SYSPRINT<sup>1</sup>
3. The Data Division glossary on SYSPRINT<sup>1</sup>
4. Formats of the object program global tables on SYSPRINT<sup>1</sup>
5. Listing of the object code on SYSPRINT<sup>1</sup>
6. A cross-reference listing on SYSPRINT<sup>1</sup>
7. An object deck on SYSPUNCH if the compiler is not running under TSO or CMS
8. The object program on SYSLIN
9. Debug data set on SYSUT5

10. FIPS messages on SYSPRINT
11. Reformatted source listing with expanded, embedded cross-referencing information on SYSPRINT
12. Reformatted source deck on SYSPUNCH
13. A verb summary listing on SYSPRINT

The user can also request a conditional or unconditional syntax-checking compilation. When unconditional syntax-checking is requested, the compiler scans the source text for syntax errors and generates the appropriate error messages, but does not generate object text. When conditional syntax-checking is requested, the compiler scans the source text for syntax errors and generates the appropriate error messages. If no message exceeds the warning (W) or conditional (C) level, a full compilation is produced. Otherwise, the object text is not generated.

"Compiler Options" in this chapter describes each of these options.

## DESIGN OF THE COMPILER

The design of the COBOL compiler is based on the structure of the source program.

In a COBOL source program, a clear distinction is made between descriptions of the operations to be performed and descriptions of the data to be operated upon. The Environment and Data Divisions provide information about the files and data items. The Procedure Division lists the operations to be performed, and specifies in what sequence and under what conditions particular operations are to be performed.

Diagram 1 (located within the Foldouts at the back of this book) illustrates how the compiler uses this distinction. Phases 10, 12, 20, 22, and 21, the data translation phases, develop the Environment and Data Divisions into areas of allocated storage for data items (including the specification of constant values where required), and provide buffers and control blocks for files. Phases 1B, 3, 35, 4, 45, 50, and 51, the procedure translation phases, break down the Procedure Division

-----  
<sup>1</sup>Output is written directly on SYSPRINT if the NOLVL option is in effect. If the LVL option is in effect, the output is written on SYSUT6, which is used by phase 80 to produce a listing on SYSPRINT.

into executable instructions. The output of these phases is used by phase 6 or phases 62, 63, and 64, the assembler phases, to produce a machine language program. The section "Compiler Phases" in this chapter discusses the function of each phase.

The phases communicate with each other by means of texts. Each phase produces a text for input to a subsequent phase. The text produced is the result of this phase's analysis of the text it received. Texts are written out on the compiler's work files as a series of elements. (An element is a logical unit of information; for example, one type of Data IC-text element is the description of a single data item.) Those texts named "IC-text" ("internal compiler") are passed from one translation phase to another and those named "A-text" ("assembler") are used as input by phase 6 or phases 62, 63, and 64 (note that P0-, P1-, P1A, ATM-, and P2-texts are forms of Procedure IC-text). A summary of the contents of each type of text is included in the description of the phase that produces it. The passing of text between phases is shown in Diagram 1.

Phases also pass information in tables. A table is a collection of information in a specified format that is left in storage. Some tables are built, used, and released all within one phase; others are passed from one phase to another. (Some tables are built by one phase for use by a subsequent phase which does not follow immediately; in this case, the table remains in storage during all the intervening phases. For example, CKPTBL built by phase 10 for phase 21: this table is resident in storage during phases 1B, 20, and 22, although not used.) The uses of these tables are described in the chapters on the phases which build and use them.

A special type of table is the dictionary. Each data-name, file-name, and procedure-name in the program is entered in the dictionary, together with all the information collected about that item. The dictionary is unique among the compiler's tables in that it is the only table which may overflow its allocated storage. If the dictionary becomes too large, direct-access data set SYSUT1 is used as a spill file to hold the overflow. After the dictionary has been built, the dictionary description of an item is written (in P1-text) in place of that item's name (in P0-text), and the dictionary is no longer needed.

Transfer of information also takes place through COMMON, or the communications area, a collection of cells of information

resident in storage throughout compilation and accessible to every phase. Each cell holds a prescribed type of information; these are listed in "Section 5. Data Areas."

The COMMON area is represented in each phase by a DSECT which describes the displacement of each field, with the same name as it exists in the first phase 00 CSECT. The actual address of COMMON is passed to the phase as a parameter of the LINK macro instruction which gives control to the phase.

#### COMPILER DATA SETS

The source module to be compiled appears as input to the compiler on the SYSIN and, optionally, the SYSLIB data sets. Direct-access data set SYSUT1 and utility data sets SYSUT2, SYSUT3, SYSUT4, SYSUT5, and SYSUT6 are used as work files. The output of compilation, depending on the options specified by the source programmer, appears on SYSPRINT, SYSLIN, SYSPUNCH, and SYSTEM. Diagram 1 shows how these data sets are used. Note that SYSUT5 is not really an internal work file, but is used to hold output from the compile step if the SYMDMP option is in effect. SYSUT6 is used only when Federal Information Processing Standard (FIPS) flagging has been requested.

#### COMPILER PHASES

The following text lists the phases in the order in which they receive control, and summarizes the functions and text output of each phase. The flow of information for all phases, except phases 00, 01, 02, and 03, is shown in Diagram 1.

#### Phase 00 (IKFCBL00)

Phase 00 is resident in storage throughout compilation. It handles most of the interfaces to the operating system by the compiler, both for receiving and returning control, and for performing input/output operations. When another phase has control, it calls phase 00 to request input/output operations and to access the compiler's tables. Phase 00 also performs interphase processing and links to the next phase.

Phase 01 (IKFCBL01)

Phase 01 contains the installation default values of compilation parameters. It passes these to phase 02.

Phase 02 (IKFCBL02)

Phase 02 performs compiler initialization. It processes the compilation parameters, determines buffer sizes, and obtains storage for buffers, tables, and the dictionary.

Phase 03 (IKFCBL03)

Phase 03 issues error messages in the event that a terminal error condition arises. This phase receives control from phase 00 and returns to phase 00, which then terminates compilation of the program and, for a batch compilation, all other programs in the batch.

Phase 04 (IKFCBL04)

Phase 04 is an optional phase. It processes the COPY statement. In addition, it performs BASIS processing. COPY allows insertion of prewritten COBOL entries, which reside in a library, into a COBOL source program at compile time. COPY also allows the user to alter these prewritten entries at compile time. If the BASIS or COPY facility is used, the LIB option must be in effect. Phase 04 reads the user-created COBOL libraries, and passes the entire source program to phases 10 or, optionally phase 12 and 1B or to phase 05 if LSTCOMP or LSTONLY is in effect.

Phase 05 (IKFCBL05)

Phase 05 is the first of three Lister phases, which are used only when the LSTCOMP or LSTONLY option is in effect. It analyzes the syntax of the COBOL source program and inserts syntactic markers between the elements of the source program.

Phase 06 (IKFCBL06)

Phase 06 is the second Lister phase. It inserts cross-reference information into the source program based on the syntactic markers provided by phase 05. During one or more passes of the file, phase 06 resolves references and merges them into the source program.

Phase 08 (IKFCBL08)

Phase 08 is the last Lister phase. It produces Lister output in accordance with the options specified. The output consists of a reformatted listing of the source program with cross-reference information; the listing begins with a preface that explains the cross-reference information that is provided by the Lister. Depending on the options in effect, phase 08 will provide a reformatted source deck and/or pass the source program to phase 10 for compilation.

Phase 10 (IKFCBL10)

Phase 10 is the first of the five data translation phases. It reads the Identification, Environment, and Data Divisions of a source program and performs syntax analysis for these divisions. From the Environment and Data Divisions, it produces Data IC-text, consisting of file and data descriptions translated into internal format. If the user requests a source program listing, phase 10 produces the listing for the Identification, Environment, and Data Divisions.

Phase 12 (IKFCBL12)

Phase 12 is the second data translation phase; it receives control only if a Report Section appears in the Data Division. It expands Report Writer statements in the Environment and Data Divisions into Report Writer subroutines, which are written out as P0-text (see "Phase 1B" below). If the user requested a source program listing, phase 12 produces the listing for the Report Section.

Phase 1B (IKFCBL1B)

Phase 1B is the first of the seven procedure translation phases. It reads the

source program Procedure Division and translates it into P0-text, one form of Procedure IC-text containing the procedure statements in a format internal to the compiler. In general, there is a one-to-one correspondence between COBOL words and P0-text codes. Data-names, file-names, and procedure-names are reproduced in P0-text in their external EBCDIC forms. A dictionary entry is made for every procedure-name. The dictionary entry contains the internal procedure-name (PN number) and attributes of the procedure-name.

Phase 20 (IKFCBL20)

Phase 20, the third data translation phase, reads Data IC-text and creates partial dictionary entries, called ATF-text. Phase 20 also produces incomplete Data A-text (see "Phase 21" below) for VALUE clauses.

Phase 22 (IKFCBL22)

Phase 22, the fourth data translation phase, reads incomplete Data IC-text and ATF-text. From these texts, it creates complete entries for LDs, CDs, and RDs (record-level descriptions) and partial (dummy) entries for SDs and FDs in the dictionary. The entry for each item is referred to by a unique dictionary pointer, which is used by later phases as the internal name of the data item. If any data items were described by the OCCURS clause with the DEPENDING ON option, phase 22 produces P0-text (see "Phase 1B" above) for special subroutines called Q-Routines, which calculate the lengths of the OCCURS...DEPENDING ON fields and the locations of the fields that follow them. DEF-text, which provides information about the point of definition of data-names and file-names required for cross-reference listings, is also created by phase 22 and passed through phase 21 to phase 6 or 64.

Phase 21 (IKFCBL21)

Phase 21, the last data translation phase, completes the SD and FD dictionary entries and the translation of Data IC-text into Data A-text. Data A-text, which is used by phase 6 or 64, provides address constants to make the data area of the object program addressable, and it specifies constant values to be placed in the data area where they are required (including fields for DCBs, DECBS, FIBs).

Phase 25 (IKFCBL25)

Phase 25 is an optional phase. It receives control only if the user requested the symbolic debug option (SYMDMP). Phase 25 builds two tables (DATATAB and OBODOTAB) on the debug data set (SYSUT5) which are used by the object-time COBOL library debugging subroutines.

Phase 3 (IKFCBL30)

Phase 3 is the second procedure translation phase. It produces P1-text, a type of Procedure IC-text, by replacing all data-names, CD-names, file-names, and procedure-names with their dictionary attributes. If a procedure statement uses the CORRESPONDING option, phase 3 breaks this statement down into several statements, each naming elementary data items. If a Data Division glossary was requested, phase 3 produces it.

Phase 35 (IKFCBL35)

Phase 35, an optional phase, is the third procedure translation phase. It processes USE FOR DEBUGGING statements and their operands. Phase 35 is invoked only if WITH DEBUGGING MODE is specified, and there are USE FOR DEBUGGING declaratives present. Phase 35 produces P1A-text to be processed by phase 04.

Phase 4 (IKFCBL40)

Phase 4, the fourth procedure translation phase, performs syntax analysis on the P1-text. Its output is P2-text, in which complex and implied verbs (such as IF, COMPUTE, CALL, PERFORM) are broken down into simpler statements, and ATM-text (which is a subset of P2-text) for the UNSTRING verb.

Phase 45 (IKFCBL45)

Phase 45, the fifth procedure translation phase, receives control only if an UNSTRING verb is encountered in the source program. It translates ATM-text from phase 4 into P2-text (see "Phase 4" above) for the UNSTRING verb.

#### Phase 50 (IKFCBL50)

Phase 50, the sixth procedure translation phase, reads P2-text. Phase 50 begins the process, which phase 51 will complete, of breaking the P2-text down into Procedure A-text -- assembler-like statements which, in general, have a one-to-one correspondence with machine instructions. Phase 50 also produces Optimization A-text, used by phase 6 or 62 to eliminate unnecessary storage duplication. The output of phase 50 is Intermediate A-text, which consists of intermediate Procedure A-text and intermediate Optimization A-text for input to phase 51; and, in the case of literal definitions, final Optimization A-text for phase 6 or 62. For the statement number option (STATE) or the symbolic debug option (SYMDMP), or the Interactive Debug facility (TEST), phase 50 generates linkages to object-time COBOL library debugging subroutines.

#### Phase 51 (IKFCBL51)

Phase 51 continues the process phase 50 began of breaking down P2-text into assembler-like statements, which in general have a one-to-one correspondence with machine instructions. The assembler-like statements are written out as Procedure A-text. Phase 51 also produces Optimization A-text, used by phase 6 or 62 to eliminate unnecessary storage duplication. For any of the debugging options (STATE, SYMDMP, FLOW, or the Interactive Debug option (TEST)), phase 51 generates linkages to object-time COBOL library debugging subroutines.

#### Phase 6 (IKFCBL60)

Phase 6 is the assembler phase. It combines all the information in the Procedure A-text, Optimization A-text, and Data A-text to produce an object program suitable for input to the linkage editor. The object program is written on SYSLIN and/or punched on SYSPUNCH according to the user's options. Phase 6 also produces an object program listing, if requested. For the statement number option (STATE), phase 6 produces Debug-text, which contains the card numbers of the source program statements, and their location within the object module.

#### Phases 62, 63, and 64 (IKFCBL62, IKFCBL63, and IKFCBL64)

Phases 62, 63, and 64 are the optional version of the assembler phase (phase 6); these phases are given control if the user requested the optimizer option (OPT), the symbolic debug option (SYMDMP), or the Interactive Debug option (TEST). Like phase 6, the function of phases 62, 63, and 64 is to produce an object program suitable for input to the linkage editor. Phase 62 reads and processes Optimization A-text and both phases 62 and 63 read Procedure A-text which phase 63 converts into Procedure A1-text. Phase 64 uses Data A-text and Procedure A1-text to complete the object (machine language) program. The object program produced by phases 62, 63, and 64 is optimized for instructions generated from the Procedure Division. The object program is written on SYSLIN and/or punched on SYSPUNCH according to the user's options. Phases 62 and 64 also produce an object program listing if requested. For the SYMDMP or the statement number option (STATE), phase 63 produces Debug-text.

#### Phase 65 (IKFCBL65)

Phase 65 is an optional phase. It receives control only if the user requested the statement number option (STATE), the symbolic debug option (SYMDMP), the flow trace option (FLOW), or the Interactive Debug option (TEST). For STATE, phase 65 uses the Debug-text written by phase 6 or 63 to produce two tables in the object module. For SYMDMP and TEST, phase 65 builds the remaining tables for the Debug data set on SYSUT5 (phase 25 has already produced two of the tables and written them on SYSUT5). For STATE, SYMDMP, or TEST these tables are used by the object-time COBOL library debugging subroutines. For FLOW, phase 65 places the number of traces requested in the variable portion of the Task Global Table in the object module. For any of the debugging options, the end card of the object module is written in phase 65. If the program is segmented and if the OPT or SYMDMP option is in effect, phase 65 copies onto SYSLIN and/or SYSPUNCH the independent segments written on SYSUT1 by phase 64.

#### Phase 6A (IKFCBL6A)

Phase 6A is an optional phase. It receives control only if the user requested a cross-reference listing by specifying the



SXREF, XREF, VBREF, and/or VBSUM option. In this case, phases 22 and 3 produced DEF-text, specifying the internal card number of the card in which every data-name, file-name, and procedure-name in the program was defined. This text was read intermixed with other text and rewritten by phase 6 or 64. Phase 6 or 64 also produced REF-text, giving the card number for every reference to a data-name, file-name, or procedure-name. Phase 6A uses these texts to write the cross-reference listing.

would have been written on SYSPRINT by previous phases if the LVL option had not been specified; phase 00 diverts all SYSPRINT output to SYSUT6 whenever the LVL option is specified. The output from phase 80 is written on SYSPRINT or SYSTEMR; the output consists of all data written on SYSUT6 with the COBOL source program flagged according to the specified FIPS level. Upon completion, phase 80 sets the return code and returns to phase 00.

Phases 70, 71, and 72 (IKFCBL70, IKFCBL71, and IKFCBL72)

Phase 70 is given control only if source program errors were detected by the translation phases (phases 10 through 51), or if the ERRMSG program-id is specified. Its input is E-text. Any phase that found an error produced an E-text element, specifying the error message to be written. These E-text elements were passed on intermixed with other text produced by the phase until control was passed to phase 51 (unless a syntax-checking compilation was requested, in which case, E-text was written on a separate data set as direct input to phase 70 by phases 4, 50, and 51). When phase 51 encountered an E-text element written by another phase, or produced an E-text element itself, it wrote the element on a separate data set (see Diagram 1 for the flow of E-text). This data set also contained E-text from the data translation phases, intermixed with Data A-text and DEF-text. In general, E-text was read and saved in storage by phase 6 or 64. Phase 70 uses the E-text to produce a list of error messages (and warning messages if the user requested them) on SYSPRINT and/or SYSTEMR. Phases 71 and 72 contain only message texts and are linked, as needed, by phase 70.

Phase 80 (IKFCBL80)

Phase 80 scans the source program for deviations from the Federal Information Processing Standard (FIPS) and issues messages along with the compiler print output, including the source program listing. Phase 80 is the last phase executed when the LVL option is in effect. Note that this phase can be optional only if NOLVL was specified as the default at installation time. Phase 80, which performs its own input and output operations, and obtains its input from SYSUT6. Input consists of all output that

COMPILER OPTIONS

The options listed below control certain events during compilation. The compiler recognizes the presence or absence of each option. The user may set default values for the options at installation time, or they may be set via the EXEC statement or the COBOL command under TSO or CMS at compilation time. The NAME option may also be set via the CBL card at compilation time. The underscore indicates the default option that is assumed by the compiler if not set otherwise.

SIZE = YYYYYYY  
indicates the amount of storage, in bytes, available for compilation.<sup>1</sup>  
This information is used by phase 02.

BUF = YYYYYYY  
indicates the amount of storage, in bytes, to be allocated to buffers.<sup>1</sup> If both SIZE and BUF are specified, the amount allocated to buffers is included in the amount of storage available for compilation. Buffers are allocated in phase 02.

SOURCE  
NOSOURCE  
indicates whether the source module is to be listed or not. This listing is produced by phases 10 and 12 for the Identification, Environment, and Data Divisions, and phase 1B for the Procedure Division. If the compiler is invoked by the COBOL Prompter under TSO, the default value is NOSOURCE. SOURCE is always in effect when LVL is specified.

LIB  
NOLIB  
indicates that the SYSLIB data set is to be opened. This information is used by phase 04 to process BASIS and COPY statements in the source program.

-----  
<sup>1</sup>The SIZE and BUF compile-time parameters can also be given in multiples of K (K=1024 bytes), for example, SIZE=128K.

If there are neither BASIS nor COPY statements in the source program, specifying NOLIB allows more efficient processing.

LOAD  
NOLOAD

indicates whether the object program is to be placed on a direct-access or a tape volume so that the program can be used as input to the linkage editor. The object program is written, if specified, by phase 6 or 62 and 64.

SYNTAX  
CSYNTAX  
NOSYNTAX  
NOCSYNTAX

indicates that a syntax-checking compilation is to be done. When SYNTAX (unconditional syntax-checking) is specified, the compiler scans the source text for syntax errors and generates the appropriate error messages, but does not generate object code. When CSYNTAX (conditional syntax-checking) is specified, the compiler scans the source text for syntax errors and generates the appropriate error messages. If no message exceeds the warning (W) or conditional (C) level, a full compilation is produced. Otherwise, the object text is not generated. When SYNTAX is specified, all of the following options are suppressed: LOAD, KREF, SXREF, CLIST, NOSUPMAP, PMAP, DECK, SYMDHP, OPT, TRUNC, FLOW, STATE, VBREF, VBSUM, COUNT, and NAME. When CSYNTAX is specified, the above options are suppressed only if one or more error (E) or disaster (D) level messages are generated. If both SYNTAX and CSYNTAX are specified, CSYNTAX overrides SYNTAX.

BATCH  
NOBATCH

indicates whether batch compilation is requested, which allows multiple programs and/or subprograms to be compiled with a single invocation of the compiler. This information is used by phases 10, 1B, and 6 or 64. BATCH should not be specified for the same compilation as SYMDHP, TEST, or LVL. If both are specified, BATCH overrides SYMDHP, TEST, or LVL.

NAME  
NONAME

indicates to phase 6 or 64 that, if the BATCH option is also specified, a linkage editor control card must be generated so that the object module will be a separate load module.

TERM  
NOTERM

indicates to phase 00 that diagnostic messages are to be directed to the SYSTEM data set as well as the SYSPRINT data set. Progress messages are also directed to the SYSTEM data set. If the compiler is invoked under CMS or by the COBOL Prompter under TSO, the default value is TERM.

NUM  
NONUM

indicates to phases 10, 12, and 1B, or 04 that line numbers recorded in columns 1-6 of the source statements are to be passed to the subsequent phases in the internal compiler text instead of compiler-generated card numbers. If a source statement number contains a nonnumeric character or appears out of sequence, the compiler generates a card number equal to the last source statement number plus 1. Hence, source statement numbers will appear in diagnostic messages and PMAP, CLIST, XREF, SXREF, READY TRACE, FLOW, SYMDHP, and STATE references rather than compiler-generated card numbers. If the compiler is invoked by the COBOL Prompter under TSO, the default value is NUM.

OPT  
NOOPT

indicates that the object module is to be optimized by phases 62, 63, and 64 for instructions generated from the Procedure Division.

FLOW=n[n] or FLOW  
NOFLOW

indicates whether the object program is to include the flow trace option. This option causes a formatted trace of the last n[n] procedures executed before an abnormal termination to be printed at execution time. If this option is specified, special P2-text elements are produced by phase 4, and phase 51 generates calls to the COBOL library flow trace subroutine. Phase 02 places the number of traces requested in COMMON. At execution time, initial control is passed to the COBOL library flow trace subroutine by the object-time COBOL library debugging control subroutine, which is called during INIT3.

STATE  
NOSTATE

indicates whether the object program is to include the statement number option, which prints the last statement number and verb number executed before the occurrence of an abnormal termination. If this option

is specified, phase 6 writes Debug-text on SYSUT2 or phase 63 writes Debug-text on SYSUT4 for use by phase 65. At execution time, control is passed to the COBOL library statement number subroutine by the object-time COBOL library debugging control subroutine. STATE should not be specified for the same compilation as SYMDMP. If both are specified, SYMDMP overrides STATE.

**SYMDMP**  
**NOSYMDMP**

indicates that a formatted symbolic dump of specified data areas is to be printed on SYSPRINT dynamically, at various points, as requested prior to execution of a program; or that, in the event of abnormal termination, a formatted symbolic dump of all data areas is to be printed on SYSPRINT. SYMDMP should not be specified for the same compilation as BATCH or STATE. If both BATCH and SYMDMP are specified, BATCH overrides SYMDMP; if both STATE and SYMDMP are specified, SYMDMP overrides STATE. If more than one COBOL program with the SYMDMP option is included in the job step and if all of the Debug data sets are on the same direct-access device, each must be given a unique name. Specification of SYMDMP automatically causes OPT to be in effect for the compilation. If WITH DEBUGGING MODE and USE FOR DEBUGGING declaratives are specified, SYMDMP will be cancelled.

**TEST**  
**NOTEST**

indicates that the program is to be executed with the TSO Interactive Symbolic Debug package. This option overrides the STATE, FLOW, COUNT, and IF WITH DEBUGGING MODE and USE FOR DEBUGGING declaratives are specified, SYMDMP will be cancelled.

**SXREF**  
**NOSXREF**  
**XREF**  
**NOXREF**

indicates that a cross-reference listing is to be produced. If the SXREF or the XREF option is specified, special text elements are produced by phases 22, 3, and 6 or 64, and phase 6A is called to generate the listing. If SXREF is specified, an alphabetically ordered cross-reference listing is generated. If XREF is specified, a cross-reference listing ordered by source statement sequence is generated. SXREF should not be specified for the same compilation as XREF. If both SXREF and XREF are specified and if the compiler was

invoked by the COBOL Prompter, SXREF overrides XREF. Otherwise, if both are specified, the last option specified is used.

**CLIST**  
**NOCLIST**

indicates that global tables, literal pool, register assignments, Working-Storage message, and a condensed listing are to be produced by phase 6 or phases 62 and 64. The procedure portion of the listing will contain only the source or compiler-generated card number, verb name, and relative location of the first instruction for each verb. CLIST should not be specified for the same compilation as PMAP. If both PMAP and CLIST are specified, PMAP overrides CLIST.

**SUPMAP**  
**NOSUPMAP**

causes phase 6 or phases 62, 63, and 64 to suppress their output if a D-level or E-level error message is generated by the compiler.

**DMAP**  
**NODMAP**

indicates whether phase 3 is to print a Data Division glossary and whether phase 6 or 62 is to print global tables, literal pool, register assignments, and the Working-Storage message.

**PMAP**  
**NOPMAP**

indicates whether global tables, literal pool, register assignments, Working-Storage message, object code listing, and assembler language expansion of the source module is to be listed by phase 6 or phases 62 and 64. PMAP should not be specified for the same compilation as CLIST. If both PMAP and CLIST are specified, the last one specified has precedence.

**ZWB**  
**NOZWB**

indicates whether the compiler is to generate code to strip the sign when comparing a signed external decimal field to an alphanumeric field. If ZWB is specified, the signed external decimal field is moved to an intermediate field and has its sign stripped before being compared to the alphanumeric field. Note that the default value cannot be changed at installation time.

**TRUNC**  
**NOTRUNC**

indicates whether standard truncation

is to be applied to computational items. If TRUNC (standard truncation) is specified and the number of digits in the sending field is greater than the number of digits in the receiving field, the computational item is truncated to the number of digits specified in the PICTURE clause of the receiving field when moved. With nonstandard truncation, they are truncated according to the actual amount of storage each item occupies. This option determines the instructions generated by phase 51.

DECK

NODECK

indicates whether the object program produced by phase 6 or by phases 62 and 64 is to be punched by phase 00 on SYSPUNCH or, under CMS, on the virtual punch. Under TSO, the object program is never punched.

RESIDENT

NORESIDENT

indicates whether the library management facility is to be used. The library management facility allows a single copy of a library subroutine to be shared by all COBOL programs in the same or different partitions or regions. If DYNAM is specified, RESIDENT is also in effect for the compilation.

DYNAM

NODYNAM

indicates whether all user subprograms are to be dynamically loaded at object time. If DYNAM is specified, RESIDENT is also in effect for the compilation.

QUOTE

APOST

indicates to phases 10, 12, and 1B, or 04 to accept either the double quotation marks (") or the apostrophe (') as the character to delineate literals and to use that character in the generation of figurative constants.

SEQ

NOSEQ

indicates whether phases 10, 12, and 1B, or 04 are to check the sequence of the source module statements. If the statements are not in sequence and SEQ is specified, a message is printed. If the compiler is invoked by the COBOL Prompter under TSO, the default value is NOSEQ. If LSTCOMP or LSTONLY is in effect, this option is ignored.

LINECNT = nn

indicates the number of lines to be printed on each page of the compiler

output listing. It is used as control information by phase 00.

FLAGW

FLAGE

indicates to phase 70 the type of messages that are to be listed for the compilation. FLAGW indicates that all warning and diagnostic messages are to be listed. FLAGE indicates that all diagnostic messages are to be listed, but not warning messages.

SPACE1

SPACE2

SPACE3

indicates to phase 00 the type of spacing to be used on the listing.

SYST

SYSx

indicates whether SYSOUT or SYSOUx (where 'x' is an alphanumeric character) is the DDname of the file to be used for data when SYSOUT is specified (implicitly or explicitly) in a DISPLAY statement. When more than one program in a job step access this file, the DDname specified in the first program is used.

VERB

NOVERB

indicates whether procedure-names and verb-names are to be listed with the associated code on the object-program listing. VERB has meaning only if the PMAP or CLIST compiler option is specified or if READY TRACE is used in the source program.

LVL = c

NOLVL

indicates whether the Federal Information Processing Standard (FIPS) flagger is to be activated. c indicates the level of the standard to be checked (A = low; B = low intermediate; C = high intermediate; D = full standard). Where LVL = c is designated as the default at installation time, NOLVL cannot be specified at compile-time.

ENDJOB

NOENDJOB

indicates whether or not, at the end of each job, COBOL library subroutines are to be called to delete modules, free storage acquired through GETMAINS issued by the COBOL program or COBOL library subroutines, close DCBs opened by subroutines and free their associated buffers. Specifying ENDJOB prevents fragmentation of storage for programs executed on the system after the COBOL program. This option takes effect at a STOP RUN statement in any

program and at a GOBACK statement in a main program only.

LSTONLY  
LSTCOMP  
NOLST

LSTONLY indicates that a listing of the reformatted source program is to be produced, but that the program is not to be compiled; if the FDECK option is in effect, the updated source deck will also be produced. LSTCOMP indicates that, in addition to the listing and optional deck produced by the LSTONLY option, the source program is to be compiled. NOLST indicates that the Lister is not to be used. When NOLST is in effect, the FDECK, CDECK, LCOL1, and LCOL2 options are ignored. The L120 and L132 options are ignored if NOLST and NOBATCH are specified.

CDECK  
NOCDECK

CDECK indicates that the updated and reformatted copy libraries are to be punched. If the FDECK option is in effect, the libraries will be punched as part of the source deck. If the NOFDECK option is in effect, the libraries will be punched as a separate deck. If NOLST is in effect, this option is ignored.

FDECK  
NOFDECK

FDECK indicates that an updated source deck is to be produced. If the CDECK option is in effect, the updated source deck will include the updated and reformatted copy libraries. If NOLST is in effect, this option is ignored.

LCOL1  
LCOL2

LCOL1 indicates that the Procedure Division is to be listed in single-column format. LCOL2 indicates that the Procedure Division is to be listed in double-column format. If NOLST is in effect, this option is ignored.

L120  
L132

L120 indicates that the print line is 120 characters long. L132 indicates that it is 132 characters long. If NOLST is in effect, this option is ignored unless BATCH is specified. If L120 or L132 is specified on an EXEC PARM card, the option is in effect for any compilation with LSTCOMP or LSTONLY in the batch compilation. The L120 or L132 option on a CBL card is ignored.

COUNT  
NOCOUNT

COUNT indicates that an execution summary is to be produced at the end of execution of the compiled program.

VBREF  
NOVBREF

VBREF indicates that a verb cross-reference listing is to be produced for the compiled program. The VBREF option also implies the VBSUM and VERB options.

VBSUM  
NOVBSUM

VBSUM indicates that a verb summary listing is to be produced for the compiled program. The VBSUM option also implies the VERB option.

DUMP  
NODUMP

DUMP indicates that the compiler will issue an ABEND for a D-level message condition caused by a possible compiler error. NODUMP indicates that the ABEND will not be issued and that the D-level message will be produced.

ADV  
NOADV

indicates whether or not records for files with WRITE...ADVANCING need reserve the first byte for the control character. ADV specifies that the first byte need not be reserved.

LANGLVL(1|2)

indicates which level of the American National Standard (ANS) COBOL definition should be used by the compiler when it encounters those few language elements whose meaning changed from 1968 to 1974. The ANS X3.23-1968 interpretation is indicated by LANGLVL(1); the ANS X3.23-1974 interpretation is indicated by LANGLVL(2), which is also the default. New language elements (those not in the 1968 ANS standard at all) are unaffected by this option. They will be accepted by the compiler even if LANGLVL(1) is specified. All IBM extensions to the language are also unaffected.

#### STORAGE REQUIREMENTS

Phase 00, which is resident in storage throughout compilation, occupies 18K bytes of storage (where K=1024 decimal). The additional storage required by each of the other phases is as follows:

<u>Phase</u>	<u>Storage (in bytes)</u>
01	236
02	14K
03	2K
04	20K
05	20K
06	4K
08	20K
10	38K
12	36K
21	26K*
22	34K*
25	10K*
3	16K**
35	8K
4	52K
45	8K
50	42K
51	50K
6	34K
62	20K
63	10K

64	26K
65	8K
6A	12K
70	28K
71	20K
72	16K
80	36K
1B	34K*
20	26K

Each phase except phases 01 and 02 uses TAMER tables and requires table space. The amount of space needed for tables varies greatly with each compilation.

-----  
 \*These phases build the dictionary and include from 2K to 3K bytes for ACCESS dictionary-handling routines.  
 \*\*An additional 1K bytes is included for ACCESS routines; the dictionary is also present during this phase.

## SECTION 2. METHOD OF OPERATION

### PHASE 00

Phase 00 (IKPCBL00), the interface between the COBOL compiler and the operating system, is resident in storage throughout compilation. Its major functions are:

1. Receiving control from the operating system and, at the end of compilation, returning control to it.
2. Performing reallocation and linkage functions after each of the other phases has completed its operations.
3. Handling input/output requests from the other phases and providing routines to handle permanent input/output errors.
4. Manipulating tables for the other phases.
5. Providing a communications area (COMMON) for the other phases.

### RECEIVING CONTROL FROM THE OPERATING SYSTEM

Compilation is invoked by an EXEC control card or by a CALL, LINK, ATTACH, or XCTL macro instruction containing the compilation parameters. Phase 00 receives the call at entry point START, and in routine LINKA links to phase 01 and passes the parameters to it for processing.

### RETURNING CONTROL TO THE OPERATING SYSTEM

When phase 4, 6, 08, 64, 65, 6A, or 70 is the last phase, it calls phase 00 to terminate compilation of the current program. If there are no more programs to compile (BATCH), phase 00 puts a code into register 15 indicating the highest source program error severity level that was encountered and branches on register 14 to return to the operating system. If an error occurs that stops compilation (see "Syntax-checking Compilations" and "Terminal Error Conditions" in this chapter), phase 00 returns to the operating

system in the same manner. When LVL is in effect, phase 80 is the final phase and it returns control to the system.

Phase 00 keeps track of which processing phase is currently active by means of a 2-byte cell named LINKCNT. Routine LINKA increments LINKCNT by 2 before it links to the next phase so that, for example, the value of LINKCNT is 2 at entry to phase 01. LINKCNT is not incremented for phases 02, 71, and 72 since the linking of these phases is transparent to phase 00. Also, there is no unique value assigned to LINKCNT for phase 03, the error handling phase.

Figure 1 traces the routines used by phase 00 for both normal and abnormal end of compilation.

### PROCESSING BETWEEN PHASES

Other phases call phase 00 for between-phase processing or for input/output request with the following sequence:

L	register,=A(COS)	See note 1.
BALR	0,register	
DC	X'XY'	See note 2.
DC	X'ZZ'	See note 3.

#### Notes:

1. A(COS) is the relocated address of the entry point to phase 00. Routine LINKPH1 passes this address in register 1 to each phase that it calls into storage.
2. 'XY' is a hexadecimal linkage code. 'X' bits indicate the function to be performed. 'Y' bits indicate the affected file. 'Y' bits are ignored when the function does not involve a file. See Figure 2 for 'XY' values.
3. 'ZZ' indicates functions to be performed by phase 00. See Figure 2 for 'ZZ' values.

Other phases call the table handling routines of phase 00 at the entry point of each routine. These routines are discussed in "Appendix A. Table and Dictionary Handling."

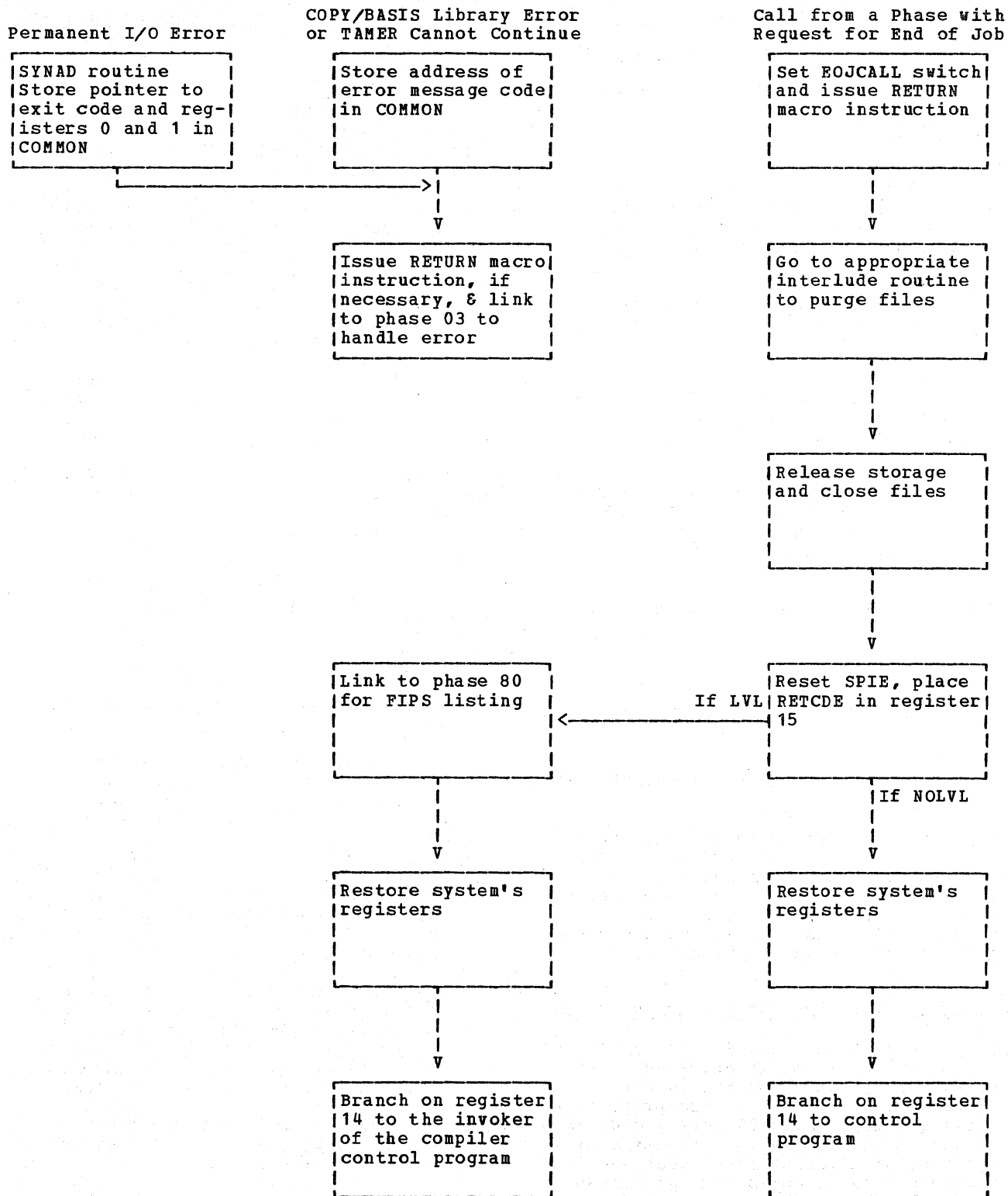


Figure 1. Flow of Control at End of Compilation



X Code	Routine Called	Function of Routine	Y Code	Data Set
0	READ	Reads a utility data set. Passes back to caller the storage address of the logical record.	1	SYSUT1
1	WRITE	PUTN: Writes a record, where caller gives address and length of data.	2	SYSUT2
2	WRITEA	PUT: Writes Data IC-text, where caller gives address of data and the first two bytes of data give length.	3	SYSUT3
3	OPENLIB	Issues OPEN to library data set, and passes back the return code to the calling phase		
6	CLOSET	Issues SVC 23 (temporary close). If a second parameter byte containing X'00' follows the 'XY' code, the data set will not be purged before TCLOSE. If a second parameter byte contains X'01', the data set will be purged before TCLOSE.	4	SYSUT4
7	READ	Reads SYSIN.	5	SYSIN
8	WOUT	Pads to the right with blanks to fill up a record for the print data set.	6	SYSPRINT
9	SEGPNT	Positions access mechanism to a disk address supplied by caller (on SYSUT1) and reads the record. See "Segmentation Operations" in this section.	7	SYSPUNCH
A	LINKB	Issues RETURN to terminate the previous processing phase.	8	SYSLIN
B	EOJ	Returns to operating system.	9	SYSLIB
C	SEGNOTE	Records the relative disk addresses of Procedure A-text on SYSUT1 for phase 51. See "Segmentation Operations" in this section.	A	SYSTEM
D	EJECT or SKIP	Positions printer. The exact function is determined by a second parameter byte as follows: X'00' eject X'01' skip 1 line X'02' skip 2 lines X'03' skip 3 lines	B	SYSUT5
E	WG01	PUT: Moves logical record into a buffer. If a second parameter byte containing X'CC' follows the 'XY' code, the call was not from phase 00 internally but from another phase.		
F	CLOSER	When a file is to be closed, moves 'FF' into buffer to indicate end-of-file, checks previous input/output operation, writes and closes the file.		

Figure 2 (Part 1 of 2). Linkage Codes to Phase 00

ZZ Code	Meaning
01	NOTE macro instruction to retrieve the absolute address.
02	POINT macro instruction to cause processing to start at the specified block in the data set.
03	POINT macro instruction to rewind the data set.
04	WRITE UPDATE (disk only).
05	When preceded by an 'XY' code of X'22', WRITE on SYSUT2 from SYSUT5 buffer with a buffer size of 512 bytes; when preceded by an 'XY' code of X'02', READ from SYSUT2 into SYSUT5 buffer with a buffer size of 512 bytes.

Figure 2 (Part 2 of 2). Linkage Codes to Phase 00

Figure 3 shows the flow of control for processing between phases.

When the calling code received by phase 00 indicates that the next phase is to be linked (X'A0'), phase 00 issues a RETURN macro instruction and then uses LINKCNT to determine which phase interlude routine to branch to.

Each interlude routine sets the PURGER string and branches to purge data sets and issue progress messages, as required. The interlude routine then performs TCLOSE, OPEN, and CLOSE operations, as necessary; makes any required changes to the buffer pointer table through which buffers are assigned to data sets; and determines which phase is to be linked next and sets LINKCNT accordingly.

Each interlude routine exits to a common routine in which the value of the COBOL space constant for the next phase is set. If the next phase is larger than the previous phase, the TAMER interlude routine is called to move tables and free main storage as necessary.

Finally, LINKCNT is incremented by 2, and control is passed to the next phase through the execution of a LINK macro instruction.

Figure 4 shows the conditions under which optional phases are called.

If the LVL option has been specified phase 00 links to phase 80 for Federal Information Processing Standard flagging.

#### PHASE INPUT/OUTPUT REQUESTS

Phase 00 translates all phase input/output requests for all phases except phase 80 into branches to data management routines or SVCs. It switches the buffer pointers in the point table if the data set is double buffered, blocks and unblocks the records, and checks to determine whether the operation is completed successfully. If necessary, it also calls TAMER to handle dictionary spill during phase processing (see the section "Table and Dictionary Handling"). Figure 5 shows the input/output requests for each phase. Figure 2 shows the linkage codes used in these requests.

When a permanent input/output error occurs in any phase except phase 80, a SYNAD routine (SYAA, SYAB, or SYAD) is called to handle the error, and phase 03, the error handling phase, is linked. Phase 03 issues a SYNADAF macro instruction to obtain the input/output error message and then issues a WTO (write-to-operator) macro instruction to print the message obtained. Finally, phase 03 sets LINKCNT to 2 more than the value assigned to phase 70 and returns to phase 00 with an end-of-job calling code (X'B0'). Compilation is abandoned via the routines described earlier in this section under "Returning Control to the Operating System."

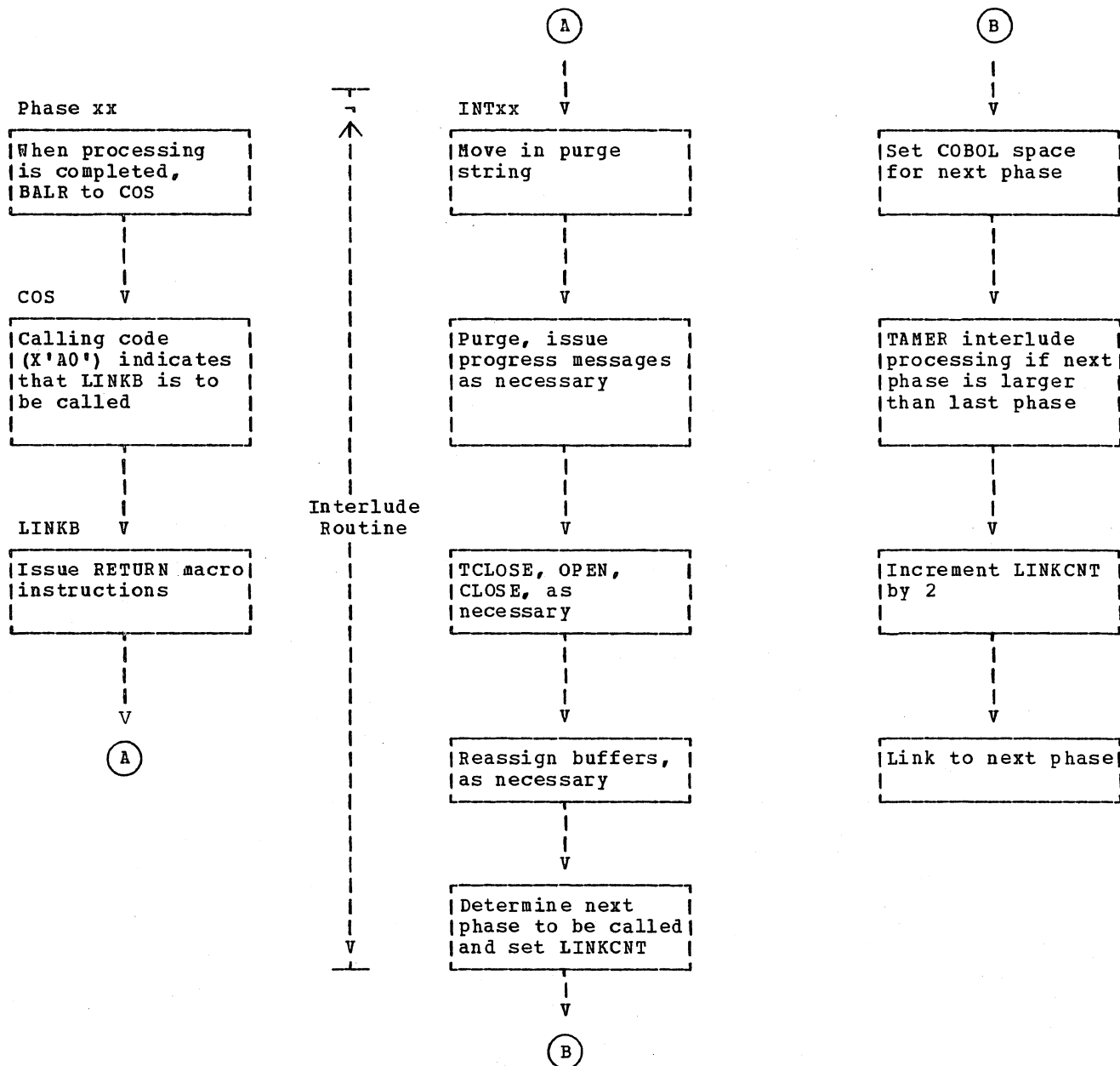


Figure 3. Flow of Control for Processing Between Phases

Optional Phase	Preceding Phase	Compiler Option
04	02	LIB
05	02	LSTONLY or LSTCOMP
06	05	LSTONLY or LSTCOMP
08	06	LSTONLY or LSTCOMP
25	21	SYMDMP or TEST
35	30	USE FOR DEBUGGING*
6A	60 64 65	XREF SXREF VBREF VBSUM
62 63 64	51	OPT
65	60 64	SYMDMP or TEST
70	Varies	If errors in source program
80	60 6A 64 65 70	LVL
*The WITH DEBUGGING MODE clause is specified under the SOURCE-COMPUTER paragraph, and the USE FOR DEBUGGING statements follow the DECLARATIVES header.		

Figure 4. Optional Phase Processing

Figure 5. Activity of the Compiler Data Sets and Buffer Assignments (Part 1 of 7)

Phase	SYSUT1 <sup>2</sup>	SYSUT2 <sup>7</sup>	SYSUT3	SYSUT4 <sup>5</sup>	SYSUT5 <sup>3</sup>	SYSIN	SYSPRINT <sup>6</sup>	SYSPUNCH	SYSLIN	SYSLIB	SYSTEM
00											WRITE
02	OPEN spill	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN WRITE	OPEN CLOSE	OPEN CLOSE		OPEN WRITE
03							OPEN				
00 INTO1 <sup>1</sup>			Purge/TCLOSE if LIB & LSTONLY or LSTCOMP Buffers 4, 5								PURGE
04 (if LIB)		WRITE/TCLOSE/READ	WRITE if LIB	WRITE Purge/TCLOSE if LIB		READ if LIB				OPEN READ	
00 INTO4			Purge/TCLOSE if LSTCOMP or LSTONLY								
05 (if LST)		WRITE if LIB	READ if LIB	READ if LIB		READ if NOLIB					
00 INTO5		Purge, TCLOSE	TCLOSE if LIB	TCLOSE if LIB							
06 (if LST)		READ WRITE	READ WRITE								
00 INTO6 <sup>6</sup>		Purge, TCLOSE	TCLOSE								
08 (if LST)		READ	WRITE	WRITE			WRITE	WRITE			
00 (if LSTCOMP) INTO8		TCLOSE	Purge, TCLOSE if LIB	Purge, TCLOSE				Purge if FDECK/CDECK			
00 (if LSTONLY) INTO8	CLOSE	CLOSE	CLOSE	CLOSE			CLOSE	Purge if FDECK/CDECK, CLOSE	CLOSE		CLOSE
10			WRITE Data IC-text E-text	READ if LIB/LSTCOMP		READ if NOLIB/NOLST	WRITE			READ	
00 INTO10		Buffers 2,3	Buffers 4,5	Buffer 1							

<sup>1</sup>Numbers specified in INTxx routines indicate the buffers used by the next phase. SYSIN, SYSPRINT, SYSPUNCH, SYSLIN, SYSLIB, and SYSTEM all use buffer 6.  
<sup>2</sup>Use of SYSUT1: Phases 1B, 22, 21, 25, and 3 use the dictionary. If dictionary spill occurs, SYSUT1 is used to write the overflow. This will never occur after phase 3 since the dictionary is released at the end of phase 3. XDAP is a macro instruction that reads what has previously been written (spilled); XDAP writes directly from the dictionary to storage and, therefore, uses no buffer. After phase 3, SYSUT1 is available for use as a utility data set.  
<sup>3</sup>Use of SYSUT5: If the SYMDMP or TEST option is not in effect, SYSUT5 is not used.  
<sup>4</sup>Reading by phase 70: Phase 70 only reads E-text from SYSUT4 if the SYNTAX option is in effect or if phase 6 or phases 62, 63, and 64 have been bypassed, that is, no reading has been done since the TCLOSE in INT51. Otherwise, phase 6 or 64 passes the E-text to phase 70 through a table in storage. If the table exceeds 256 bytes, phase 6 or 64 writes the E-text on SYSUT3.  
<sup>5</sup>Use of SYSUT4: Phase 04 writes on SYSUT4 and phases 10 and 1B read from SYSUT4 only if the LIB option is in effect. If the LIB option was not specified, that is, BASIS or COPY statements are not present in the source program, phases 10 and 1B read from SYSIN.  
<sup>6</sup>Use of SYSUT6: The SYSUT6 DDname replaces SYSPRINT when LVL is specified. After phase 80 processing, FIPS flagger output is directed to SYSPRINT or SYSTEM.  
<sup>7</sup>Use of SYSUT2: Phase 04 outputs source programs to SYSUT2 as BASIS control cards are processed. Phase 04 then reads SYSUT2 in order to process any COPY statements.

Figure 5. Activity of the Compiler Data Sets and Buffer Assignments (Part 2 of 7)

Phase	SYSUT1 <sup>2</sup>	SYSUT2	SYSUT3	SYSUT4 <sup>5</sup>	SYSUT5 <sup>3</sup>	SYSIN	SYSPRINT <sup>6</sup>	SYSPUNCH	SYSLIN	SYSLIB	SYSTEM
12 (Report Writer)		WRITE P0-text E-text	WRITE Data IC-text	READ if LIB/ LSTCOMP		READ if NOLIB/ NOLST	WRITE			READ	
00 INT12		Buffers 2,3									
1B <sup>2</sup>	WRITE XDAP	WRITE P0-text E-text		READ if LIB/ LSTCOMP		READ if NOLIB/ NOLST	WRITE			READ	
00 INT1B			Purge TCLOSE Buffers 4,5	TCLOSE Buffers 1,6			Purge				
20 <sup>2</sup>			READ Data IC-text E-text	WRITE Data IC-text Incomplete Data A-text ATF-text E-text							
00 INT20		Buffers 2,3	TCLOSE Buffers 4,5	Purge TCLOSE Buffers 1,6							
22 <sup>2</sup>	WRITE XDAP	WRITE P0-text for Q-Routines	WRITE Data IC-text Data A-text DEF-text E-text	READ Data IC-text Incomplete Data A-text ATF-text E-text							
00 INT22		Buffers 2,3	Purge TCLOSE Buffers 4,5	TCLOSE Buffers 1,6							
21 <sup>2</sup>	WRITE XDAP	WRITE P0-text	READ Data IC-text Data A-text DEF-text E-text	WRITE Data A-text E-text DEF-text							
00 INT21		Purge TCLOSE Buffers 2,3	TCLOSE	Purge Buffers 1,6	Buffer 4						
If SYMDMP 25 <sup>2</sup>			READ DEF-text	WRITE E-text	WRITE DATATAB OBODOTAB						

<sup>1</sup>Numbers specified in INTxx routines indicate the buffers used by the next phase. SYSIN, SYSPRINT, SYSPUNCH, SYSLIN, SYSLIB, and SYSTEM all use buffer 6.  
<sup>2</sup>Use of SYSUT1: Phases 1B, 22, 21, 25, and 3 use the dictionary. If dictionary spill occurs, SYSUT1 is used to write the overflow. This will never occur after phase 3 since the dictionary is released at the end of phase 3. XDAP is a macro instruction that reads what has previously been written (spilled); XDAP writes directly from the dictionary to storage and, therefore, uses no buffer. After phase 3, SYSUT1 is available for use as a utility data set.  
<sup>3</sup>Use of SYSUT5: If the SYMDMP or TEST option is not in effect, SYSUT5 is not used.  
<sup>4</sup>Reading by phase 70: Phase 70 only reads E-text from SYSUT4 if the syntax option is in effect or if phase 6 or phases 62, 63, and 64 have been bypassed, that is, no reading has been done since the TCLOSE in INT51. Otherwise, phase 6 or 64 passes the E-text to phase 70 through a table in storage. If the table exceeds 256 bytes, phase 6 or 64 writes the E-text on SYSUT3.  
<sup>5</sup>Use of SYSUT4: Phase 04 writes on SYSUT4 and phases 10 and 1B read from SYSUT4 only if the LIB option is in effect. If the LIB option was not specified, that is, BASIS or COPY statements are not present in the source program, phases 10 and 1B read from SYSIN.  
<sup>6</sup>Use of SYSUT6: The SYSUT6 DDname replaces SYSPRINT when LVL is specified. After phase 80 processing, FIPS flagger output is directed to SYSPRINT or SYSTEM.

Figure 5. Activity of the Compiler Data Sets and Buffer Assignments (Part 3 of 7)

Phase	SYSUT1 <sup>2</sup>	SYSUT2	SYSUT3	SYSUT4 <sup>5</sup>	SYSUT5 <sup>3</sup>	SYSIN	SYSPRINT <sup>6</sup>	SYSPUNCH	SYSLIN	SYSLIB	SYSTEM
00 INT25		Buffers 2,3	TCLOSE Buffers 4,5	Buffer 1							
3 <sup>2</sup>	WRITE XDAP	READ P0-text E-text	WRITE P1-text E-text	WRITE DEF-text			WRITE				
00 INT3	Purge CLOSE spill OPEN utility Buffers 1,6 or 1 if (C)SYNTAX	TCLOSE Buffers 2,3	Purge TCLOSE Buffers 4,5	Purge Buffer 6 if (C)SYNTAX		CLOSE	Purge			CLOSE	WRITE Purge
4	WRITE P2-text E-text	WRITE ATM-text READ P1-text E-text (if V2BUGDCL)	READ P1-text E-text WRITE ATM-text (if V2BUGDCL)	WRITE E-text if (C)SYNTAX							
00 INT4 No SYNTAX No CSYNTAX No UNSTRING	Purge TCLOSE Buffers 1,6	TCLOSE (if V2BUGDCL) Buffers 2,3	TCLOSE (if V2BUGDCL) Buffers 4,5								
00 INT4 CSYNTAX No SYNTAX No UNSTRING	Purge TCLOSE Buffer 1	Buffers 2,3	TCLOSE Buffers 4,5	Purge Buffer 6							
00 INT4 UNSTRING No SYNTAX No CSYNTAX	Purge Buffers 1,6	Purge TCLOSE Buffers 2,3	Purge (if V2BUGDCL) TCLOSE Buffers 4,5								
00 INT4 CSYNTAX UNSTRING No SYNTAX	Purge Buffer 7	Purge TCLOSE Buffers 2,3	Purge (if V2BUGDCL) TCLOSE Buffers 4,5	Purge Buffer 6							
35		WRITE P1A-text E-text	READ P1-text E-text								
00 INT35		Purge TCLOSE	TCLOSE								

<sup>1</sup>Numbers specified in INTxx routines indicate the buffers used by the next phase. SYSIN, SYSPRINT, SYSPUNCH, SYSLIN, SYSLIB, and SYSTEM all use buffer 6.  
<sup>2</sup>Use of SYSUT1: Phases 1B, 22, 21, 25, and 3 use the dictionary. If dictionary spill occurs, SYSUT1 is used to write the overflow. This will never occur after phase 3 since the dictionary is released at the end of phase 3. XDAP is a macro instruction that reads what has previously been written (spilled); XDAP writes directly from the dictionary to storage and, therefore, uses no buffer. After phase 3, SYSUT1 is available for use as a utility data set.  
<sup>3</sup>Use of SYSUT5: If the SYMDMP or TEST option is not in effect, SYSUT5 is not used.  
<sup>4</sup>Reading by phase 70: Phase 70 only reads E-text from SYSUT4 if the syntax option is in effect or if phase 6 or phases 62, 63, and 64 have been bypassed, that is, no reading has been done since the TCLOSE in INT51. Otherwise, phase 6 or 64 passes the E-text to phase 70 through a table in storage. If the table exceeds 256 bytes, phase 6 or 64 writes the E-text on SYSUT3.  
<sup>5</sup>Use of SYSUT4: Phase 02 writes on SYSUT4 and phases 10 and 1B read from SYSUT4 only if the LIB option is in effect. If the LIB option was not specified, that is, BASIS or COPY statements are not present in the source program, phases 10 and 1B read from SYSIN.  
<sup>6</sup>Use of SYSUT6: The SYSUT6 DDname replaces SYSPRINT when LVL is specified. After phase 80 processing, FIPS flagger output is directed to SYSPRINT or SYSTEM.

Figure 5. Activity of the Compiler Data Sets and Buffer Assignments (Part 4 of 7)

Phase	SYSUT1 <sup>2</sup>	SYSUT2	SYSUT3	SYSUT4 <sup>5</sup>	SYSUT5 <sup>3</sup>	SYSIN	SYSPRINT <sup>6</sup>	SYSPUNCH	SYSLIN	SYSLIB	SYSTEM
00 INT4 SYNTAX (Phase 70 next)				Purge TCLOSE Buffer 6							
45 (UNSTRING)	WRITE P2-text E-text	READ ATM-text (if not V2BUGDLL)	READ ATM-text (if not V2BUGDLL)	WRITE E-text if (C)SYNTAX							
00 INT45 SYNTAX (Phase 70 is next)				Purge TCLOSE Buffer 6							
00 INT45 CSYNTAX	Purge TCLOSE Buffer 1	TCLOSE (if not V2BUGDLL) Buffers 2,3	TCLOSE if V2BUGDCL Buffers 4,5	Purge Buffer 6							
00 INT45 No SYNTAX No CSYNTAX	Purge TCLOSE Buffers 1,6	TCLOSE (if not V2BUGDLL) Buffers 2,3	TCLOSE (if V2BUGDCL) Buffers 4,5	Purge							
50	READ P2-text E-text	WRITE Intermediate Procedure A-text Intermediate Optimization A-text Intermediate E-text P2-text	WRITE Final Optimization A-text	WRITE E-text if (C)SYNTAX							
00 INT50 SYNTAX (Phase 70 is next)				Purge TCLOSE Buffers 3,4							
00 INT50 No SYNTAX	TCLOSE Buffers 4,6	Purge TCLOSE Buffers 2,3	Buffer 5	Buffer 1							

<sup>1</sup>Numbers specified in INTxx routines indicate the buffers used by the next phase. SYSIN, SYSPRINT, SYSPUNCH, SYSLIN, SYSLIB, and SYSTEM all use buffer 6.  
<sup>2</sup>Use of SYSUT1: Phases 1B, 22, 21, 25, and 3 use the dictionary. If dictionary spill occurs, SYSUT1 is used to write the overflow. This will never occur after phase 3 since the dictionary is released at the end of phase 3. XDAP is a macro instruction that reads what has previously been written (spilled); XDAP writes directly from the dictionary to storage and, therefore, uses no buffer. After phase 3, SYSUT1 is available for use as a utility data set.  
<sup>3</sup>Use of SYSUT5: If the SYMDMP or TEST option is not in effect, SYSUT5 is not used.  
<sup>4</sup>Reading by phase 70: Phase 70 only reads E-text from SYSUT4 if the syntax option is in effect or if phase 6 or phases 62, 63, and 64 have been bypassed, that is, no reading has been done since the TCLOSE in INT51. Otherwise, phase 6 or 64 passes the E-text to phase 70 through a table in storage. If the table exceeds 256 bytes, phase 6 or 64 writes the E-text on SYSUT3.  
<sup>5</sup>Use of SYSUT4: Phase 04 writes on SYSUT4 and phases 10 and 1B read from SYSUT4 only if the LIB option is in effect. If the LIB option was not specified, that is, BASIS or COPY statements are not present in the source program, phases 10 and 1B read from SYSIN.  
<sup>6</sup>Use of SYSUT6: The SYSUT6 DDname replaces SYSPRINT when LVL is specified. After phase 80 processing, FIPS flagger output is directed to SYSPRINT or SYSTEM.



Figure 5. Activity of the Compiler Data Sets and Buffer Assignments (Part 5 of 7)

Phase	SYSUT1 <sup>2</sup>	SYSUT2	SYSUT3	SYSUT4 <sup>5</sup>	SYSUT5 <sup>3</sup>	SYSIN	SYSPRINT <sup>6</sup>	SYSPUNCH	SYSLIN	SYSLIB	SYSTEM
51	WRITE Procedure A-text	READ Intermediate Procedure A-text Intermediate Optimization A-text Intermediate E-text P2-text	WRITE Final Optimization A-text	WRITE E-text							
00 INT51 (Phase 60 is next)	Purge TCLOSE Buffers 4,3	TCLOSE Buffer 2	Purge TCLOSE Buffer 5	Purge TCLOSE Buffer 1							
00 INT51 OPT (Phase 62 is next)	Purge TCLOSE Buffers 1,2	Purge TCLOSE Buffers 3,4	Purge								
00 INT51 SYNTAX (Phase 70 is next)			Purge TCLOSE Buffers 3,4								
6	READ Procedure A-text  TCLOSE  WRITE DEF-text	WRITE Debug-text	READ Optimization A-text  TCLOSE  WRITE E-text if ERRTBL overflows, REF-text	READ Data A-text E-text DEF-text			WRITE	WRITE	WRITE Object Module		WRITE
00 INT60 No Phase 65 No Phase 6A			Purge TCLOSE Buffers 1,2	TCLOSE Buffers 3,4			Purge	Purge	Purge		WRITE Purge
00 INT60 Phase 65 No Phase 6A	Buffers 4,3	Purge TCLOSE Buffers 2,5	Purge TCLOSE	TCLOSE Buffer 1			Purge				WRITE Purge
<p><sup>1</sup> Numbers specified in INTxx routines indicate the buffers used by the next phase. SYSIN, SYSPRINT, SYSPUNCH, SYSLIN, SYSLIB, and SYSTEM all use buffer 6.</p> <p><sup>2</sup> Use of SYSUT1: Phases 1B, 22, 21, 25, and 3 use the dictionary. If dictionary spill occurs, SYSUT1 is used to write the overflow. This will never occur after phase 3 since the dictionary is released at the end of phase 3. XDAP is a macro instruction that reads what has previously been written (spilled); XDAP writes directly from the dictionary to storage and, therefore, uses no buffer. After phase 3, SYSUT1 is available for use as a utility data set.</p> <p><sup>3</sup> Use of SYSUT5: If the SYMDMP or TEST option is not in effect, SYSUT5 is not used.</p> <p><sup>4</sup> Reading by phase 70: Phase 70 only reads E-text from SYSUT4 if the syntax option is in effect or if phase 6 or phases 62, 63, and 64 have been bypassed, that is, no reading has been done since the TCLOSE in INT51. Otherwise, phase 6 or 64 passes the E-text to phase 70 through a table in storage. If the table exceeds 256 bytes, phase 6 or 64 writes the E-text on SYSUT3.</p> <p><sup>5</sup> Use of SYSUT4: Phase 04 writes on SYSUT4 and phases 10 and 1B read from SYSUT4 only if the LIB option is in effect. If the LIB option was not specified, that is, BASIS or COPY statements are not present in the source program, phases 10 and 1B read from SYSIN.</p> <p><sup>6</sup> Use of SYSUT6: The SYSUT6 DDname replaces SYSPRINT when LVL is specified. After phase 80 processing, FIPS flagger output is directed to SYSPRINT or SYSTEM.</p>											

Figure 5. Activity of the Compiler Data Sets and Buffer Assignments (Part 6 of 7)

Phase	SYSUT1 <sup>2</sup>	SYSUT2	SYSUT3	SYSUT4 <sup>5</sup>	SYSUT5 <sup>3</sup>	SYSIN	SYSPRINT <sup>6</sup>	SYSPUNCH	SYSLIN	SYSLIB	SYSTEM
00 INT60 Phase 65 Phase 6A	Purge Buffers 4,3	Purge TCLOSE Buffers 2,5	Purge TCLOSE	TCLOSE Buffer 1			Purge				WRITE Purge
00 INT60 Phase 6A No Phase 65	Purge TCLOSE Buffers 1,2		Purge TCLOSE Buffers 3,4	TCLOSE			Purge	Purge	Purge		WRITE Purge
If OPT 62	READ Procedure A-text		READ Optimization A-text					WRITE	WRITE		
00 INT62	TCLOSE Buffers 1,2	TCLOSE Buffers 3,4		Buffers 5				Purge	Purge		
63	READ Procedure A-text	WRITE Procedure A1-text		WRITE Debug-text			WRITE				
00 INT63	TCLOSE Buffer 1	Purge TCLOSE Buffers 2,3	TCLOSE Buffer 4	Purge TCLOSE Buffer 5							
64	WRITE DEF-text, independent segments if SYMDMP	READ Procedure A1-text	WRITE E-text if ERRTBL overflows, REF-text	READ Data A-text DEF-text E-text				WRITE	WRITE		
00 INT64 (Phase 65 is next)	Purge TCLOSE Buffer 1	TCLOSE	Purge TCLOSE	Buffers 2,3			Purge	Purge	Purge		WRITE Purge
00 INT64 (Phase 6A is next)	Purge TCLOSE Buffers 1,2	TCLOSE	Purge TCLOSE Buffers 3,4				Purge	Purge	Purge		WRITE Purge
00 INT64 (Phase 70 is next)	Purge TCLOSE	TCLOSE	Purge TCLOSE Buffers 1,2	Buffers 3,4			Purge	Purge	Purge		WRITE Purge

<sup>1</sup> Numbers specified in INTxx routines indicate the buffers used by the next phase. SYSIN, SYSPRINT, SYSPUNCH, SYSLIN, SYSLIB, and SYSTEM all use buffer 6.  
<sup>2</sup> Use of SYSUT1: Phases 1B, 22, 21, 25, and 3 use the dictionary. If dictionary spill occurs, SYSUT1 is used to write the overflow. This will never occur after phase 3 since the dictionary is released at the end of phase 3. XDAP is a macro instruction that reads what has previously been written (spilled); XDAP writes directly from the dictionary to storage and, therefore, uses no buffer. After phase 3, SYSUT1 is available for use as a utility data set.  
<sup>3</sup> Use of SYSUT5: If the SYMDMP or TEST option is not in effect, SYSUT5 is not used.  
<sup>4</sup> Reading by phase 70: Phase 70 only reads E-text from SYSUT4 if the syntax option is in effect or if phase 6 or phases 62, 63, and 64 have been bypassed, that is, no reading has been done since the TCLOSE in INT51. Otherwise, phase 6 or 64 passes the E-text to phase 70 through a table in storage. If the table exceeds 256 bytes, phase 6 or 64 writes the E-text on SYSUT3.  
<sup>5</sup> Use of SYSUT4: Phase 04 writes on SYSUT4 and phases 10 and 1B read from SYSUT4 only if the LIB option is in effect. If the LIB option was not specified, that is, BASIS or COPY statements are not present in the source program, phases 10 and 1B read from SYSIN.  
<sup>6</sup> Use of SYSUT6: The SYSUT6 DDnames replaces SYSPRINT when LVL is specified. After phase 80 processing, FIPS flagger output is directed to SYSPRINT or SYSTEM.

Phase	SYSUT1 <sup>2</sup>	SYSUT2	SYSUT3	SYSUT4 <sup>5</sup>	SYSUT5 <sup>3</sup>	SYSIN	SYSPRINT <sup>6</sup>	SYSPUNCH	SYSLIN	SYSLIB	SYSTEM	SYSUT6
If FLOW STATE or SYMDMP 65	READ Independent segments if SYMDMP	WRITE-READ if SYSUT5 is assigned to tape device		READ Debug-text	WRITE DEBUG tables for SYMDMP only			WRITE	WRITE			
If no SXREF, XREF, 00 INT65			TCLOSE if OPT Buffers 1,2					Purge	Purge			
If SXREF or XREF 00 INT65	TCLOSE Buffers 1,2		TCLOSE Buffers 3,4					Purge	Purge			
If SXREF or XREF 6A	READ DEF-text		READ REF-text				WRITE				WRITE	
00 INT6A			TCLOSE if OPT Buffers 1,2	Buffers 3,4								
70			READ E-text	READ <sup>4</sup> E-text			WRITE				WRITE	
00 INT70	CLOSE	CLOSE	CLOSE	CLOSE			CLOSE	CLOSE	CLOSE		CLOSE	
80	OPEN WRITE READ CLOSE						OPEN WRITE CLOSE				OPEN WRITE CLOSE if TERM	OPEN READ CLOSE

<sup>1</sup>Numbers specified in INTxx routines indicate the buffers used by the next phase. SYSIN, SYSPRINT, SYSPUNCH, SYSLIN, SYSLIB, and SYSTEM all use buffer 6.  
<sup>2</sup>Use of SYSUT1: Phases 1B, 22, 21, 25, and 3 use the dictionary. If dictionary spill occurs, SYSUT1 is used to write the overflow. This will never occur after phase 3 since the dictionary is released at the end of phase 3. XDAP is a macro instruction that reads what has previously been written (spilled); XDAP writes directly from the dictionary to storage and, therefore, uses no buffer. After phase 3, SYSUT1 is available for use as a utility data set.  
<sup>3</sup>Use of SYSUT5: If the SYMDMP or TEST option is not in effect, SYSUT5 is not used.  
<sup>4</sup>Reading by phase 70: Phase 70 only reads E-text from SYSUT4 if the syntax option is in effect or if phase 6 or phases 62, 63, and 64 have been bypassed, that is, no reading has been done since the TCLOSE in INT51. Otherwise, phase 6 or 64 passes the E-text to phase 70 through a table in storage. If the table exceeds 256 bytes, phase 6 or 64 writes the E-text on SYSUT3.  
<sup>5</sup>Use of SYSUT4: Phase 04 writes on SYSUT4 and phases 10 and 1B read from SYSUT4 only if the LIB option is in effect. If the LIB option was not specified, that is, BASIS or COPY statements are not present in the source program, phases 10 and 1B read from SYSIN.  
<sup>6</sup>Use of SYSUT6: The SYSUT6DDname replaces SYSPRINT when LVL is specified. After phase 80 processing, FIPS flagger output is directed to SYSPRINT or SYSTEM.

Figure 5. Activity of the Compiler Data Sets and Buffer Assignments (Part 7 of 7)

### Compiler Linkage to Data Management

Phase 00 does not use data management macro instructions for OPEN, CLOSE, TCLOSE, READ, and WRITE. To open files, phase 00 issues SVC 19; to close files, SVC 20; to close temporarily, SVC 23. To read and write, phase 00 updates the appropriate fields of the DCB and DECB, and then branches and links to the data management READ/WRITE routine. The address of this data management routine is picked up from the DCB.

The interlude (INTxx) routines, which handle opening and closing of files, issue one SVC for all files to be opened or closed at a given time. Register 1 points to a list which gives the address of DCBs for all files to be included in this SVC. The last entry in the list has the high-order bit turned on to indicate end-of-list.

### Directing Error Messages and Progress Messages to the SYSTEM Data Set

If the TERM option is in effect, error messages are directed to SYSTEM as well as SYSPRINT (SYSUT6 for LVL option). However, if SYSPRINT cannot be opened or is a dummy data set, error messages are directed to SYSTEM only.

In addition, if the TERM option is in effect, a progress message is written to SYSTEM (never SYSPRINT) by phase 00. The progress message states: "Release 2.0 OS/VS COBOL IN PROGRESS." If errors have occurred, a message stating the number of errors and the highest severity code encountered during the compilation is also written.

### Segmentation Operations

When a segmented program is being compiled, phase 00 issues NOTE and POINT macro instructions for phases 51, 6, 62, and 63. Phase 51 keeps track of the sections of Procedure A-text that belong to each segment; phase 6 or phases 62 and 63 use the information passed to them by phase 51 to construct the object module, segment by segment.

In the P2-text processed by phase 51, a segmentation control break signals a section of text whose priority is different from that of the section just processed. SEGSAVE contains the relative track and

cylinder numbers on SYSUT1 of the first record in the section. Phase 51 then calls the COS routine with a C1 in the XY parameters (see "Receiving Control from Another Phase" in this section for the meaning of X and Y). In response to these parameters, the COS routine branches to the NOTE routine, which issues a NOTE macro instruction to retrieve the relative address on SYSUT1 of the record phase 00 is about to write (that is, for the next section). The NOTE routine places the address in SEGSAVE, and phase 00 returns to phase 51.

Using the priority numbers in the SEGTBL table entries, phase 6 or phases 62 and 63 first process all the sections for one segment, then all the sections for the next, and so on. Each time it picks up the SEGTBL entry for a new section of Procedure A-text, phase 6 or phases 62 and 63 call the COS routine with a value of 91 in the XY parameters. It also passes the relative track and cylinder for the new section in its SEGTBL table entry. In response to these parameters, the COS routine branches to the SEGPNT routine, which issues a POINT macro instruction to the SYSUT1 address of the section. It then reads the first record of the new section for phase 6 or phases 62 and 63. Subsequent calls to read records of the same section are made by phase 6 or phases 62 and 63 with a value of 01 in the XY parameters.

When the OPT option is in effect, phase 62 reads, in order of ascending priority, the sections of Procedure A-text that belong to each segment to determine the main storage requirements for the program.

### TABLE AND DICTIONARY HANDLING

A portion of storage is reserved throughout compilation for tables built and used by the phases. All processing involving these tables (inserting new entries, releasing a table when no longer needed, etc.) is handled by the group of routines known collectively as TAMER. These routines are resident in phase 00. They are described in "Appendix A: Table and Dictionary Handling."

### COMMUNICATIONS AREA (COMMON)

The communications area (COMMON) is resident in phase 00. It contains information to which all phases can refer directly. The format of COMMON is given in "Communications Area" in "Section 5. Data Areas."

SYNTAX-CHECKING COMPILATIONS

An unconditional syntax-checking (SYNTAX) compilation or the production of an error (E) or disaster (D) level message during a conditional syntax-checking (CSYNTAX) compilation causes the compiler to produce no object code and to print out the appropriate messages on SYSPRINT (SYSUT6 for LVL option) and/or SYSTEM. The CSYNTAX compilation, upon generating an E or D level message, becomes in effect a SYNTAX compilation and the SYNTAX switch in the PHZSH3 cell in COMMON is set on. After phase 22, any phase detecting a syntax error sets the ERRSEV cell in COMMON to the highest error severity level encountered.

After phase 4, E-text is on SYSUT4. During INT4, phase 00 checks the SYNTAX switch. If it is on, phase 00 sets the value of LINKCNT to indicate that phase 70 is to be executed next. The RDERRFIL bit in the SWITCH cell in COMMON is set on to indicate to phase 70 that E-text is to be read from SYSUT4. After phase 70 processing, phase 00 either gives control to phase 80 if the LVL option is in effect or returns to the operating system via the routines described earlier in this section under "Returning Control to the Operating System."

If the SYNTAX switch is off, processing continues with phase 50. Phase 50 writes E-text on SYSUT4 if the CSYNTAX option is in effect; the generation of an E level message causes the SYNTAX switch to be set on. During INT50, phase 00 performs the same processing if the SYNTAX switch is on as it does during INT4. If the SYNTAX switch is off, processing continues with phase 51.

-----  
 \*If the TERM option is in effect and the permanent input/output error did not occur on a write to SYSTEM, the message is written to SYSTEM as well as the console. If the permanent input/output error occurred on a write to SYSTEM and not SYSPRINT, the message is written to SYSPRINT (SYSUT6 for LVL option) as well as the console.

Phase 51 processing for the CSYNTAX option is the same as phase 50 processing. If phase 51 detects no syntax error which would cause an E level message, a full compilation is produced.

TERMINAL ERROR CONDITIONS

The following conditions will cause compilation to be abandoned. In each case, if NODUMP is specified, phase 03 is called to print an error message on the console or SYSPRINT (SYSUT6 for LVL option); phase 03 then returns to phase 00, which returns to the operating system via the routines described earlier in this section under "Returning Control to the Operating System."

1. A permanent input/output error is encountered on a device.\*
2. An invalid COPY or BASIS library name is encountered.
3. TAMER cannot continue:
  - o Larger region needed.
  - o Compiler error.\*\*
  - o A table has exceeded the maximum permissible size.\*\*
  - o Fragmented storage.\*\*

-----  
 \*\*If the DUMP option is in effect, this error causes an ABEND with a dump.

## PHASE 01

Phase 01 (IKFCBL01) is logically a part of phase 00, but is a separate load module because it is not required in storage throughout compilation. The functions of phase 01 are:

1. To contain the installation default values of compilation parameters in a module separate from the actual initialization coding (phase 02 contains the actual initialization coding).
2. To pass these default values of the compilation parameters, together with any values of the compilation parameters chosen by the user at compilation time, to phase 02.

### COMPILATION PARAMETERS

When phase 00 links to phase 01, it passes the address of a parameter list that points to the compilation options, altered DD names, and augmented headings. The section "Introduction" in this publication explains the options that can be used on an EXEC

control card or that can be passed by the COBOL Prompter to the compiler if specified as operands of the COBOL command under TSO. These options can also be used in an ATTACH, LINK, CALL, or XCTL macro instruction. In addition, the macro instruction parameters can specify a change in the user data set names or an addition to the standard page heading for the output listing. (The standard page heading consists of a page number only.) Phase 01 passes these parameters, together with their installation default values, to phase 02.

### RETURN FROM PHASE 02

Phase 01 is a macro instruction. When phase 02 has finished processing, it returns to phase 01, thereby restoring phase 01 registers and releasing phase 02 storage. Phase 01 branches to phase 00. Phase 00 then issues a RETURN macro instruction for phase 01, which restores phase 00 registers and releases phase 01 storage.

PHASE 02

Phase 02 (IKFCBI02) is the initialization phase. It is logically part of phase 01, which contains the installation default values of compilation parameters, but phase 02 is a separate load module. The major functions of phase 02 are:

- Processing the compilation parameters specified on the EXEC card or passed with the CALL, LINK, XCTL, or ATTACH macro instruction. (If the COBOL command is used to invoke the compiler, a LINK macro instruction is executed to pass control to the compiler.)
- Determining buffer sizes for the compiler data sets for all phases.
- Obtaining storage for tables, the dictionary, and buffers.
- Entering information in COMMON to be used for statistics in the program listing.
- Opening all required data sets to check that they can be opened and to determine the block sizes specified by the user.
- Processing for the BATCH option.
- Building the table of COBOL space constants in phase 00, using the BLDL macro instruction which allows dynamic calculation of phase sizes.

COMPILATION PARAMETERS

Phase 02 sets switches in COMMON locations SWITV2, LISTERSW, PHZSW, PHZSW1, PHZSW2, PHZSW3, and PHZSW4 for phase 00 to indicate which of the following options were chosen:

ADV		
BATCH	LIB	SOURCE
CLIST	LOAD	STATE
COUNT	LSTCOMP	SUPMAP
CSYNTAX	LSTONLY	SXREF
DECK	LVL	SYMDMP
DMAP	L120	SYNTAX
DUMP		
DYNAM	L132	SYSx
ENDJOB	NAME	TERM
FLAG	NUM	TEST
FLOW	OPT	VBREF
LANGLVL(112)		
LCOL1	PNAP	VBSUM
LCOL2	QUOTE	VERB
LCPY	RESIDENT	XREF
FDECK	SEQ	ZWB

From the BUF and SIZE parameters, phase 02 determines the amount of space that is available (see "Buffer Size Determination" below). From the SPACE parameter, phase 02 sets the print control character both in the carriage control field for SYSPRINT and also in the COMMON location SPACING. From the LINECNT parameter, phase 02 records in COMMON location CNTLINE the value for the number of lines per page to be printed in the source card listing.

If augmented page headings were given in a macro instruction, phase 02 records them for phase 00 so that it can set up the proper heading print format.

If the data set names were changed by a LINK, ATTACH, or XCTL macro instruction, phase 02 changes the names in the DCB's.

Phase 02 uses the control program to determine the date of compilation, which it records in COMMON.

BUFFER SIZE DETERMINATION

The compiler uses six buffer areas. Figure 5 shows, for each phase, the buffers used by the data sets active in that phase. Because buffers 1 through 5 are always used for utility data sets, they are of uniform size so that they can be used for different data sets from phase to phase. Buffer 6 is also used for a utility data set in phases 20, 22, 21, 25, 50, and 51; therefore, it must be at least as large as the other buffers. Buffer 6 may need to be larger than the other buffers since in phases 10, 12, 1B, 3, and 6 it is used for up to three double-buffered data sets. Note that in Figure 4 if the two buffer numbers are the same, the data set is single-buffered.

Note: If the BATCH option has been specified, a seventh buffer is used. This buffer is used only by the SYSIN data set and is present throughout the compilation. It is necessary to save the input cards of subsequent compilations within a batch.

Phase 02 determines the total buffer size available from the BUF option on the EXEC control card or in the COBOL command string. If BUF was not specified but SIZE was, phase 02 calculates BUF from the formula:

BUF = 1/4 (SIZE - 96K) + 4K

If neither BUF nor SIZE was specified on the EXEC control card or in the COBOL command string, phase 02 sets BUF equal to the default value specified at installation time.

BUF is the total amount of storage available for buffers. To determine the buffer sizes from this value, phase 02:

1. Determines the block sizes of the user data sets by opening the data sets; or, if the block sizes have not been given on the DD cards or have been specified incorrectly, assigns the following default sizes:
  - 80 bytes for SYSIN, SYSLIN, SYSPUNCH, and SYSLIB.
  - 121 bytes (133 if L132 is in effect) for SYSPRINT (SYSUT6 for LVL option) and SYSTEM.
2. Using these block sizes, determines the required size of buffer 6 in phases 10, 12, 1B, 3, and 6, and chooses the larger of these sizes as the size of buffer 6. (If, for example, the default block sizes were assigned, the size of buffer 6 would be 804 bytes.)
3. Subtracts the size of buffer 6 from BUF and divides the remainder equally among the other five buffers.
4. Compares the size of buffer 6 to buffers 1 through 5. If buffer 6 is smaller, it makes all the buffers the same size, that is, one-sixth of BUF.
5. Finally, to ensure that the determined buffer size is not greater than that which the input/output devices can handle, the maximum block size permitted on the device is determined via a DEVTYPE macro instruction and compared with the calculated buffer sizes. The smaller of these two is chosen as the buffer size.

Phase 02 records the buffer sizes in phase 00 buffer control blocks. Phase 00 uses the buffer control blocks to keep track of how much of the buffer has been used in blocking and unblocking records.

#### PROCESSING FOR THE BATCH OPTION

If the BATCH option has been specified and if the compilation is the first within the batch, phase 02 reads until it encounters a

card which is not a CBL card (COBOL options card). Phase 02 then processes the options contained on all the CBL cards and alters switch settings in SWITV2, LISTERSW, PHZSW, PHZSW1, PHZSW2, and PHZSW3 in COMMON. Phase 02 also saves the address of the first non-CBL card in COMMON location ADDR CARD and sets the BATCHSW switch in COMMON to hexadecimal '08' (CARDHELD). For subsequent compilations in the batch, the COMMON location ADDR CARD contains the address of the next CBL card. After the first compilation, phase 02 tests the BATCHSW switch for a hexadecimal '08' and obtains the address of the CBL card from ADDR CARD and processes the options as described above. In addition, areas in phase 00 are initialized to their original values for subsequent compilations.

#### ENTERING STATISTICAL INFORMATION IN COMMON

In addition to the information stored in COMMON as a result of compiler option parameters, phase 02 stores other information in COMMON that it has determined during storage and buffer allocation. The total area to be used for buffers is stored in BUFSIZE. The total area available for compilation is stored in CORESIZE. Phase 00 uses the information in these cells, as well as the information in PHZSW, PHZSW1, PHZSW2, PHZSW3, CNTLINE, and SPACING (see "Compilation Parameters") to produce statistical data for the listing.

#### PREPARING FOR FEDERAL INFORMATION PROCESSING STANDARD (FIPS) FLAGGING

When the LVL option is specified, phase 02 enters the level character in the FIPLVL cell in COMMON to indicate to phase 80 what level of the FIPS standard is to be flagged (A = low; B = low intermediate; C = high intermediate; D = full standard). Phase 02 also:

- Sets the LIST bit in COMMON to indicate that the source option is in effect.
- Replaces the SYSPRINT DDname with the SYSUT6 DDname. (This data set receives the source listing that is used for input to phase 80 for FIPS flagging.)
- Opens the DCB for SYSUT6, or, if the OPEN is unsuccessful, cancels the LVL option.
- Moves the SYSUT6 DDname (or its alternate DDname) and the DDname for the FIPS output file into the DDNTBL table.



INFORMATION RETURNED TO PHASE 00

When phase 02 has finished processing, it returns to phase 01, restoring phase 01 registers and releasing phase 02 storage. Phase 01 then branches to phase 00.

Phase 02 passes the following information back to phase 00:

1. DCBs modified by the compilation parameters.
2. The switches indicating the compilation options.
3. The LINECNT indication.
4. An indication of the difference between the maximum amount of storage requested for tables, the dictionary, and buffers, and the amount actually received.
5. The address of the buffer area and the buffer lengths.
6. The date of the compilation.

ERROR CONDITIONS

Phase 02 detects and handles the following error conditions:

1. A data set cannot be opened. If the data set is SYSLIB, when required for any compiler data set, compilation is terminated. If the data set is SYSIN, SYSUT1, SYSUT2, SYSUT3, or SYSUT4, a disaster message is written and compilation is terminated. If the data set is SYSUT5, a warning message is written and the symbolic debug (SYHDMP) option is canceled. If the

data set is SYSLIN, when the LOAD option is requested, a warning message is written, the LOAD option is canceled, and compilation continues. If the data set is SYSPUNCH, when the DECK option is requested, a warning message is written, the DECK option is canceled, and compilation continues. If the data set is SYSUT6 when the LVL option is specified, a warning message is written and the LVL option is canceled. If the data set is SYSPRINT, when the NOTERM option is requested, a disaster message is written to the console and compilation is terminated. However, if SYSTEM can be opened when SYSPRINT (SYSUT6 for LVL option) cannot be opened, a warning message is written to the console and compilation continues. If the data set is SYSTEM when the TERM option is requested, phase 02 checks to see if SYSPRINT is usable. If SYSPRINT can be opened and it is not a dummy data set, a warning message is written to both SYSPRINT and the console, the TERM option is canceled, and compilation continues. However, if the TERM option is requested but SYSTEM cannot be opened and SYSPRINT cannot be opened or is a dummy data set, a disaster message is written to the console and compilation is terminated.

2. Specified storage device block size is larger than the buffer space allocated, or is not an even multiple of the record length. The block size is set to equal the length of one logical record, and a message indicating this is printed.
3. Invalid or insufficient SIZE or BUF parameter. Warning only: an alternate value is chosen.

PHASE 03

Phase 03 (IKFCBL03) is logically part of phase 00, but is a separate load module because it is required to be in storage only if a terminal error condition occurs (see "Terminal Error Conditions" in the section "Phase 00"). Phase 03 is linked to by phase 00 whenever a SYNAD routine exit is to be taken, when TAMER cannot continue, or when phase 00 encounters an invalid COPY or BASIS library name.

The functions of phase 03 are to issue error messages and to indicate to phase 00 that compilation is to be terminated.

OBTAINING AND PRINTING ERROR MESSAGES

Before linking to phase 03, phase 00 places the address of a one-byte error code in the KTRMNAE cell in COMMON. It also saves the contents of registers 0 and 1 in the SYNADRO1 location in COMMON, these are needed for the SYNADAF macro instruction. The following codes are used:

Code (hexadecimal)	Meaning
00	SYNAD exit for SYSLIB
01	SYNAD exit for all data sets except SYSLIB and SYSTEM
02	SYNAD exit for SYSTEM
06	Larger region needed
07	Compiler error
08	A table has exceeded the maximum permissible size
09	Fragmented storage
0A	Invalid BASIS request
0B	Invalid library name

-----  
 \*If the DUMP option is specified, phase 00 issues an abend with dump instead of linking to phase 03.

Phase 03 examines the code. If a SYNAD exit is indicated, registers 0 and 1 are loaded from COMMON and a SYNADAF macro instruction is issued to obtain the input/output error message. This message with a COBOL prefix is then written out on the console through the execution of a WTO (write-to-operator) macro instruction.

Phase 03 next calls phase 00 to print out the message on SYSPRINT and/or SYSTEM according to the following conditions. If the error occurred on the SYSTEM data set, the message is written out on SYSPRINT if the SOURCE option is in effect. If the error occurred on the SYSLIB data set, the message is written out on SYSPRINT if the SOURCE option is in effect and on SYSTEM if the TERM option is in effect. If the error occurred on a data set other than SYSLIB or SYSTEM, the message is not written out on SYSPRINT, but is written out on SYSTEM if the TERM option is in effect.

If a SYNAD exit is not to be taken, phase 03 obtains the text and length of the appropriate message and then calls phase 00 to print out the message on SYSPRINT and/or SYSTEM depending upon whether the SOURCE and/or TERM option is in effect.

RETURNING CONTROL TO PHASE 00

After printing the error message, phase 00 returns to phase 03. Phase 03 sets LINKCNT to 2 more than the value assigned to phase 70 and sets the BATCHSW switch in COMMON to indicate that compilation of other programs in the batch is to be bypassed. Finally, phase 03 returns to phase 00 with an end-of-job (X'BO') calling code.

PHASE 04

Phase 04 (IKFCBL04) implements the COPY language of American National Standard COBOL, X3.23-1974, by allowing insertion of prewritten COBOL entries, which reside in a library, into a COBOL source program at compile time. COPY also allows the option to alter these prewritten entries at compile time.

Phase 04 reads the user-created COBOL libraries and passes the entire source program to phases 10 and 1B, or to phase 05 if LSTCOMP or LSTONLY is in effect. The LIB option must be in effect if the BASIS or COPY facility is used.

Input

The input to phase 04 can be source input text and library text.

Output

During BASIS processing SYSUT2 is used as output during BASIS processing. This is then read as source to process any COPY statements. SYSUT4 is used as final output from phase 04. If Lister is not to be invoked, a source listing is produced on SYSPRINT.

ERROR CONDITIONS

Phase 04 contains text for error messages and produces the error messages. It produces E-text on SYSUT3 as it scans the source program listings and analyzes syntax.

TABLES USED

Phase 04 creates the DCBTBL which contains one entry for each unique library name. Upon completion of processing, phase 04 deletes DCBTBL.

MAIN FLOW OF CONTROL IN PHASE 04

Phase 04 invokes the PH04INIT subroutine to initialize variables, allocate work area storage and build the DCBTBL, then reads the initial SYSIN record.

If BASIS is desired, phase 04 invokes the BASISRTN subroutine to read and update the BASIS library from SYSLIB. The resulting source is written to SYSUT2.

Phase 04 next invokes the COPYRTN subroutine to scan SYSIN or SYSUT2 (if BASIS) source for COPY statements. COPYRTN invokes COPYROC to syntax check a COPY statement and to read and update the source from the identified COPY library member.

When source end-of-file is reached, phase 04 closes any COPY libraries that have been opened.

PROCESSING ROUTINES USED

BASISRTN: The BASISRTN routine merges source from the BASIS library specified with the users source from SYSIN, matching serial numbers appearing on the BASIS INSERT and DELETE cards with those in columns 1 through 6 on the BASIS library source. When the merge is complete, SYSUT2 is TCLOSED and made ready for COPYRTN processing, and SYSLIB is closed.

COPYRTN: The COPYRTN routine reads all source input looking for a COPY verb that is not part of a NOTE, comment line, or comment entry. Whenever COPY is found, COPYRTN invokes COPYROC to process the statements. When source end-of-file is reached, COPYRTN returns to the phase 04 controller routine for phase termination.

COPYPROC: COPYROC validates the COPY statement.

The library is identified and opened. The REPLACING arguments are passed and saved in a work area. Any syntax or OPEN errors will result in the COPY statement being nullified. If the COPY statement has no errors, the COPY library is read and the source updated according to the REPLACING rules. At library end-of-file, COPYROC returns control to COPYRTN.

PHASE 05

Control is given to phase 05 (IKFCBL05) only when the Lister option (LSTONLY or LSTCOMP) has been specified. Phase 05 is the Lister scan phase, which analyzes the syntax of the COBOL source program. This phase inserts syntactic markers between the various elements of the source program. The syntactic markers are used by subsequent phases to produce the cross-references and to reformat the program for the Lister option listing.

Phase 05 is divided into major functions that:

- tokenize the output (COBOL source) into syntactic units or words (the SCAN and READ routines)
- decode Y instructions from the YBGN table
- place COBOL source and inserted syntactic markers into a large body of contiguous storage known as the HOLDAREA (in order that delayed recognition of COBOL statements can occur)
- output COBOL source and syntactic markers in the correct sequence from the HOLDAREA to SYSUT2

Input

If NOLIB is in effect, the input to phase 05 is the COBOL source program. Input can be read from the card reader or the system input device. If COPY/BASIS is in effect, the input is passed from phase 04 on SYSUT4 (source with COPY/BASIS resolved) and on SYSUT3 (E-text).

Output

The output from phase 05 is written on SYSUT2. The output consists of the COBOL source program with syntactic markers inserted to identify the various elements of the program. Syntactic markers indicate such items as new statement, reference type, level number, indentation, and qualifiers. If phase 05 detects syntax errors, the output also includes error and recovery markers, to indicate that the errors are to be identified in the Lister

output listing. Also, any E-text read from SYSUT3 is written on SYSUT2.

SYNTAX LANGUAGE SUMMARY

The syntax analysis done by Phase 05 is a table-driven process whereby the table entries (called Y-instructions in the Phase 05 assembler listing) control the recognition of COBOL source and resulting generation of phase 05 output.

$[O] \left\{ \begin{array}{l} W \\ P \\ T \end{array} \right\} [A[B]] \cdot \left\{ \begin{array}{l} \text{Reserved word} \\ \text{Punctuation} \\ \text{Operand term} \end{array} \right\}$ <p>(see Note 3)</p> <p style="text-align: center;">• Clause name</p>	<p>G[n] * Text-type byte M[n] * Modifying byte C n *</p> <p>(see Note 5)</p> <p>X * Exit routine name</p>
$[O] [N] [1] / [ \left\{ \begin{array}{l} \text{number of items in clause} \\ \text{name of first statement beyond} \end{array} \right\} ]$ <p>(see Note 4)</p>	

Notes:

1. Syntax language is written as a sequence of statements, optionally named, each of which contains a sequence of items separated by commas.
2. Syntax language items perform tests, define clauses, or control generation. These functions are denoted by an infixed period, slash, or asterisk, respectively.
3. Tests may be:
  - O optional (brackets)
  - W testing reserved words
  - P punctuation
  - T generic terms

For example, DATANAME, ALPHALIT, or "clauses" (nul), such as "identifier" (IDFR), and test may specify A margin (A), B margin (B), or both (AB).

4. Clause definition corresponds to the COBOL use of brackets, braces, and elipses:

0 is optional (brackets)  
 N may be repeated (elipses)  
 1 select only one (braces)

Information after the slash specifies end-of-clause; if omitted, the end of the current statement is assumed.

5. Generation (of IPTEXT) is implied by successful tests, and is explicitly ordered by G, the type-byte implies the length. M modifies one byte of already generated text. C changes an index to the current point of generation for possible subsequent use by M or C commands. X causes execution of an exit routine.

6. Items in general return results in quaternary logic. "True" means a positive identification, "false" means a clear failure, "maybe" results from generation or from a failed optional test, and "disaster" results from a "true" test followed by a "false" within a clause where all items must be found.

ERROR CONDITIONS

Phase 05 terminates upon detecting a syntax error that it detects in the COBOL source program. When such an error is detected, phase 05 issues an error flag to signal phase 08 that the following source cards are to be passed on without processing. Phase 05 then treats the balance of the program as comment cards.

In addition to the condition mentioned above, unusual termination of phase 05 can occur if the source program contains:

- Too many (approximately 80 or more) consecutive \*-comments cards.
- Too many (approximately 100 or more) consecutive blank cards.

If one of the above two conditions (of phase 05) occurs, the file written on SYSUT2 is incomplete.

## PHASE 06

Phase 06 (IKFCBL06), the Lister sort phase, inserts cross-reference information into the source program. Phase 06 makes two or more passes of the file created by phase 05. Based on the syntactic markers contained in the file, this phase inserts pointers into the source program as follows:

- At the place of definition of a name, pointers to the places where references to that name occur.
- At the places of reference, pointers to the place of definition.

During each pass of the file, phase 06 resolves references and merges them into the source program; the number of passes depends on the amount of storage available and the number of cross-references to be processed. A partial dictionary of all definitions is used by all passes. The dictionary is continually updated by adding new definitions as space becomes available and deleting definitions that have been completely processed and are no longer needed.

### Input

The input for the first pass of phase 06 is the file written on SYSUT2 by phase 05. Input for subsequent passes of phase 06 is the output of the previous pass. That is, input is read alternately from SYSUT2 and SYSUT3 (beginning with SYSUT2).

### Output

The output of phase 06 is written alternately on SYSUT3 and SYSUT2. Output of the first pass of phase 06 is always written on SYSUT3 and output of the last pass is always written on SYSUT2. The output file consists of the source program with cross-reference information embedded in it; the contents of the file are printed by phase 08.

The functions of phase 08 (IKFCBL08) are as follows:

- Print the first page (preface) of the Lister listing
- Print the body of the Lister listing
- Depending on the options specified
  - Punch the reformatted source program deck on SYSPUNCH
  - Pass the reformatted source program, via SYSUT4 to phase 10 for compilation
  - Write COPY/BASIS-related E-text on SYSUT3

#### Input

For the preface, phase 08 uses no input.

For the remainder of the listing, phase 08 reads input from SYSUT2. Input consists of the source program with embedded cross-reference information from phase 06.

#### Output

The output of phase 08 consists of:

- The preface which describes
  - The format of the listing
  - The use of statement numbers
  - The classification of references
  - The use of footnotes in the listing
  - The method of indentation
  - The reformatted deck that can be obtained
  - The summary listing
- The Lister option listing

- An internal card-image COBOL source program
- A reformatted source deck.

The Lister option preface and listing are printed on SYSPRINT. The internal card-image source program, which may serve as input for subsequent compilation, is produced on SYSUT4 if the LSTCOMP option is in effect. The reformatted source deck is produced on SYSPUNCH if the Lister FDECK option is in effect.

#### Processing

From the source program with embedded cross-reference information, phase 08 builds an entire page in storage. The phase reformats the source program and creates footnotes as required. When the optimum place for a new page is reached, phase 08 prints the created page on SYSPRINT and then deletes the page from storage. The process is repeated until all data from SYSUT2 has been processed. To produce the summary listing, phase 08 positions SYSUT2 at the beginning and reads it again.

#### ERROR CONDITIONS

It is possible that some footnotes on some COBOL programs may be lost. If a particular COBOL program requires a very large number of footnotes, there may not be enough storage space to contain the complete footnote table. In those cases, some of the footnotes that could be printed in the Procedure Division of the Lister will be handled as ordinary data references, rather than as footnotes.

Upon encountering the error flag, phase 08 issues a message informing the user that Lister processing has terminated because of a source syntax error. The balance of the program is then printed (and passed on SYSUT4, if LSTCOMP) without reformatting or cross-referencing).

PHASE 10

Phase 10 (IKFCBL10) reads the source statements in the Identification, Environment, and Data Divisions. As it reads the card images of the source program, it performs the following major functions:

1. Storing information from the Environment and Data Divisions in the form of Data IC-text and as entries of tables supplementing the text (see "Section 5. Data Areas" for formats).
2. Sorting all other significant information in other tables and in cells of the COMMON communications area (see "Section 5. Data Areas" for format).
3. Analyzing the syntax of source statements.
4. Writing each statement on SYSPRINT as part of the source program listing, if the user specified the SOURCE option.
5. Checking for the CBL card, if the user specified the BATCH option. When the CBL card is found, phase 10 sets on the hexadecimal '08' (CARDHELD) and hexadecimal '40' (ENDFOUND) values in the BATCHSW switch in COMMON and stores the address of the CBL card in COMMON location ADDR CARD.

These functions are performed under the control of three major working routines and three division-processing routines. Among the working routines, GETDLM supervises the division processors and GETWD and GETCRD supply them with input. Among the division processors, IDDCSN processes the Identification Division, ENVSCN the Environment Division, and DDSCN the Data Division.

Syntax analysis, although not specifically described here, is performed as part of division processing. It includes such functions as checking for division headers and the proper position of words and clauses. If user errors are detected, the division processors call routines of the generic name MSGxxx (where xxx is the message number) to generate E-text.

MAJOR WORKING ROUTINES

There are three routines in phase 10 that are used extensively by more than one of the division processing routines. These are the GETDLM, GETWD, and GETCRD routines.

GETDLM ROUTINE

The major functions of the GETDLM routine are:

- Searching for delimiters (division headers, level numbers, etc.) and passing control to the proper division routines.
- Recognizing literals that are level numbers and encoding them as such.
- Causing termination of phase 10 when it recognizes the Report Section, the Procedure Division header, or end-of-file on the input device.

GETWD ROUTINE

The major functions of the GETWD routine are:

- Getting a logical unit from the input card provided by the GETCRD routine, identifying and encoding it, and sending it to the calling routines for processing. A logical unit is defined as all the characters between one blank and the next.
- If the NUM compiler option is not in effect, generating a card number for each input card, starting with 1 and (except for Report Writer statements) incrementing by 1. If the NUM compiler option is in effect, the user-specified card number is converted to binary. In either case, the current card number is kept in a three-byte cell labeled CURGCN and in the COMMON cell CURCRD. If the STATE, TEST, or SYMDMP option and the NUM option are in effect, a warning (W) level message is issued if card numbers are found to be out of sequence. Beginning with the first out-of-sequence card number, phase 10 resequences the card numbers,



incrementing by one, until, and unless, the source card numbers fall within the ascending sequence again.

- Ensuring that the next logical unit is valid for the division being processed.

Each logical unit is analyzed and encoded into internal phase 10 code which tells the processing routine what type of item it is (COBOL word, qualified EBCDIC name, etc.).

#### GETCRD ROUTINE

The GETCRD routine gets the next card image according to the user's options and writes a line on SYSPRINT if the SOURCE option was requested. If the LIB or LSTCOMP option is in effect, the source program is read from SYSUT4.

The GETWD and GETCRD routines are also used by phases 1B and 12.

#### IDENTIFICATION DIVISION

The Identification Division scan routine (IDDSCN) is entered immediately after the phase 10 initialization routines. The input is scanned for an Identification Division header. When it is encountered, the cell for the next logical unit is filled by the GETWD routine and checked to determine whether it is PROGRAM-ID. If it is, the program-name is saved in the PROGID cell of COMMON (see "Section 5. Data Areas") to be used later either as the CSECT name of the object module or to form the names of segments in a segmented program.

After the PROGRAM-ID (if any) has been saved and if the SOURCE option was specified, the Identification Division is written on SYSPRINT. If DATE-COMPILED is included, the information that follows it is deleted, and the current date is inserted from COMMON.

When another division header is encountered by routine GETDLM, control passes to the proper division scan routine.

#### ENVIRONMENT DIVISION

When the Environment Division header is encountered, the Environment Division scan routine (ENVSCN) searches for the Configuration and Input-Output Sections and

then branches to the routines that process them. These routines produce the file definition portion of Data IC-text (see in "Section 5. Data Areas"), which will be combined with the data definition portion later in phase 10.

#### CONFIGURATION SECTION

The OBJECT-COMPUTER paragraph, including the SEGMENT-LIMIT clause, and the SPECIAL-NAMES paragraph are processed.

SOURCE-COMPUTER Paragraph: If the WITH DEBUGGING MODE clause is specified, the V2BUGON switch in COMMON will be set on.

OBJECT-COMPUTER Paragraph: If the computer-name specified is IBM-370 [-model-number], the S370IN switch in the PH1BYTE cell in COMMON is set to 1.

If PROGRAM COLLATING SEQUENCE, save the name to be checked for during SPECIAL-NAMES processing.

SEGMENT-LIMIT Clause: When the priority number specified is less than 50, this number is stored in the SEGLMT cell of COMMON. If no SEGMENT-LIMIT clause is specified, or if the value exceeds 49, SEGLMT is set to 49.

SPECIAL-NAMES Paragraph: The SPNSCN routine processes the SPECIAL-NAMES paragraph.

CURRENCY-SIGN Clause: The literal specified is checked for validity (based on the setting of LANGLVL) and then stored in the CURSGN cell of COMMON. Thereafter, whenever phase 20 scans the PICTURE clause, it recognizes the literal as the currency sign.

DECIMAL-POINT Clause: A code is entered in the COMMD cell of COMMON. Thereafter, when phases 10, 12, and 1B scan numeric and floating-point literals, commas instead of periods are recognized as decimal points.

System-name Is Mnemonic-name: An entry is made in the SPNTBL table for each mnemonic-name, and in the ALPHTRL for each alphabet-name. These table entries are used by phase 1B in processing the Procedure Division. When phase 1B scans ACCEPT, DISPLAY, or WRITE statements, it replaces any mnemonic-names with the proper system-name by checking the SPNTBL table. When it processes any SORT or MERGE verb with COLLATING SEQUENCE specified, it checks the ALPHTRL for a valid name.

INPUT-OUTPUT SECTION

The routines that process the Input-Output Section build and use the ENVTBL and QNMTBL tables. The QNMTBL is a table of variable-length names, with pointers to each of its entries in the appropriate fields of the ENVTBL entries. These tables are released later in phase 10. Formats of these tables are given in "Section 5. Data Areas."

FILE-CONTROL Paragraph

The SELSCN routine processes the SELECT clauses and produces partial Data IC-text (see "Section 5. Data Areas") from the information obtained. At the end of Environment Division processing, the text is in the form of ENVTBL entries, one for each SELECT clause in the source program, and contains only file information. Subsequent Data Division processing uses the ENVTBL entries for the completion of Data IC-text (see "File Section" in this chapter). The text itself is translated into Data A-text in phase 21 and is further processed in phase 6 or 64.

For each SELECT clause, the file-name and other pertinent information are entered in the ENVTBL table. Variable-length names are entered in QNMTBL; pointers to the QNMTBL entries are placed in the appropriate ENVTBL fields.

For FILE STATUS clause processing, SELSCN passes control to the file status routine to enter the FILE STATUS dataname into the QNMTBL and to set a pointer to QNMTBL in the corresponding ENVTBL table field. A corresponding bit is also turned on in the ENVTBL to indicate that a FILE STATUS clause has been specified.

For PASSWORD clause processing, SELSCN passes control to the password routine to enter the password into the QNMTBL and to set a pointer to QNMTBL in the corresponding ENVTBL table field. A corresponding bit is also turned on in the ENVTBL to indicate that a PASSWORD clause has been specified.

The ALTSCN routine gains control from SAENT whenever ALTERNATE is found in a SELECT statement. ALTSCN first tests the FRSTALT bit switch, checking for ALTERNATE RECORD KEY under current SELECT. For all ALTERNATE RECORD KEY clauses, CHKSAR enters the data-name in QNMTBL. If the DUPLICATES option is present, a bit is turned on in INDTBL for that key. When another ALTERNATE clause is found in NXCOD, a

chain bit is turned on in the INDTBL entry. Control is returned to SAENT after FRSTALT is reset.

I-O-CONTROL Paragraph

When the ENVSCN routine encounters the I-O-CONTROL paragraph header, it calls the pertinent routines for processing the SAME, RERUN, and APPLY clauses.

SAME Clause: For each clause encountered, the files named are entered into one of the following tables:

<u>Clause</u>	<u>Table</u>
SAME AREA	SATBL
SAME RECORD/AREA	SRATBL

At the end of Environment Division processing, a unique number is assigned to each clause of each type (by means of incrementing counters). For example, the first SAME AREA clause is assigned the number 1, the second SAME AREA clause is assigned 2, and so forth. Similarly, the first SAME RECORD AREA clause is assigned 1, the next clause 2, and so forth.

The ENVTBL table is then searched for all the files named in SAME clauses, and appropriate numbers are inserted into these ENVTBL entries to identify the SAME clauses in which the files are named. For example, if three SAME RECORD AREA clauses were specified, each file named in the first clause would have "1" in the SAME RECORD AREA field of its ENVTBL entry: each file named in the second SAME RECORD AREA clause would have "2" in that field, and so forth. The appropriate switches are also set in the ENVTBL entries. At this time, the SATBL and SRATBL tables are released and may be overlaid by new tables.

RERUN Clause:

Format 1: An entry for the file named is made in the CKPTBL table and a bit is set in the entry to indicate whether the "END OF REEL/UNIT" or the "integer-1 RECORDS" option was specified in the RERUN clause. Both options can be applied to one file, and, in this case, two entries are made in the CKPTBL table. In the ENVTBL entry for the file, the CKPTBL bit is set to 1 and a pointer to the first of two possible entries in the CKPTBL table is inserted. A pointer to the second entry, if specified, is entered in the first entry in the CKPTBL table. Later when the FD statement for this file is scanned, the CKPTBL bit is tested. If it is 1 and if the "integer-1 RECORDS" option was specified in the RERUN clause, the RERUN

bit in the PIOTBL table entry for the file is set to 1. Phase 1B uses this bit in processing READ and WRITE statements.

Format 2 (SORT-RERUN): An entry is made in the CKPTBL table, with the INTEGER field set to 0. The RERUNN switch in COMMON is set.

APPLY Clause: For each option, a switch is set in the ENVTBL entry for the file-name specified in the clause.

Option	Switch Set
1	WRITE-ONLY
2	CORE-INDEX
3	RECORD-OVERFLOW
4	REORG-CRITERIA

Note that the data-name is entered in the QNMTBL table, and a pointer to the QNMTBL entry is placed in the ENVTBL entry for this file.

MULTIPLE FILE TAPE Clause: clause is treated as comments.

#### DATA DIVISION

When the Data Division header is encountered, the GETDLH routine calls the DDSCN routine, which in turn calls the routines that process the File, Working-Storage, Linkage, and Communication Sections. If a Report Section is encountered, phase 00 is called to load phase 12, which processes the Report Section.

As the Data Division source statements are encountered, the following steps are taken to form Data IC-text:

1. File and record information is entered into a work area called ICTEXT.
2. Entries are made in the OD2TBL, TOTTL, QNMTBL, FNTBL, and RCDTBL tables.
3. Information in the work area is merged with the corresponding ENVTBL entry (for file descriptions).
4. Data IC-text for FDS, LDs, SDs, and CDs is generated and written on SYSUT3. At this time, space is reserved in the PIOTBL table.

Completed Data IC-text is used in phases 22 and 21 to make dictionary entries for data-names and file-names and to generate Data A-text.

Data Division processing uses the ENVTBL table and builds the FNTBL, OD2TBL, PIOTBL, QNMTBL, RCDTBL, and TOTTL tables. All of these tables (see "Section 5. Data Areas" for formats) except the ENVTBL table are passed to phase 1B. The PIOTBL table is also used by phase 21, the OD2TBL table by phases 22 and 25, and the TOTTL table by phase 22. The PIOTBL table indicates which OPEN options and input/output verbs are used for each file; and the OD2TBL table is used to generate Q-routines (object module subroutines used to calculate variable lengths and locations for OCCURS...DEPENDING ON fields).

#### FILE SECTION

When the File Section header is encountered, the DDSCN routine calls the appropriate routines to process FDS, SDs, and LDs in the source program.

#### File Description Entries

Each File Description (FD) entry is analyzed, and information from the clauses is entered in the work area ICTEXT. (The REPORT clause requires special processing; it is described in the chapter "Phase 12.") A blank PIOTBL entry is created, and pointers to this entry are placed in the FNTBL and ENVTBL tables. The file-name from the FD is entered in the FNTBL table entry and the LABEL RECORDS switches are set. The ACCESS RANDOM and mass-storage switches are picked up from the ENVTBL entry. Variable-length names such as LABEL RECORD names are entered in the QNMTBL, and pointers to these entries are placed in the work area.

When this processing has been completed, the ENVTBL entry and the file description information from the work area are merged into Data IC-text elements. Names from the QNMTBL table are located (from their ENVTBL pointers) and inserted where needed. Each completed Data IC-text element is written out. When File Section processing has been completed, the QNMTBL table is released.

#### Sort Description Entries

Each Sort Description (SD) entry is placed in a work area and is used to generate an SD entry in Data IC-text.

Record Description Entries

When a level number is encountered signaling a record description entry, the entry is analyzed and information from the clauses is put into a work area. If the OCCURS...DEPENDING ON clause is included, the object of the clause and its qualifiers are entered in the OD2TBL (no duplicate entries are made). A pointer to the entry is inserted later into the Data IC-text for the record description. (Each level number results in a Data IC-text element in LD-text format; see "Internal Text Formats.")

For level-01 items, an RCDTBL entry is made, consisting of a pointer to the most recent file-name entry in the FNTBL, followed by the record-name. This results in the arrangement of pointers shown in Figure 6. The relationships are used by phase 1B when processing WRITE and REWRITE statements to relate records to their associated files.

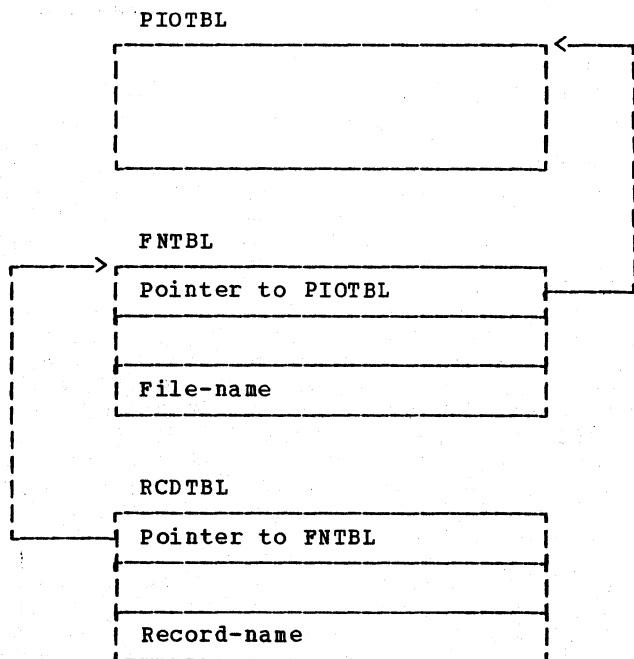


Figure 6. Table Usage During Record Description Processing

Then the LD-text is written out.

WORKING-STORAGE AND LINKAGE SECTIONS

The record descriptions in the Working-Storage and Linkage Sections are processed in much the same way as those in the File Section. However, since they are not associated with files, no RCDTBL entries are made.

COMMUNICATION SECTION

When the Communication Section header is encountered, the DDSCN routine calls the CDSCNA routine to process the CDs in the source program. Each Communication Description (CD) entry is analyzed and Data IC-text elements are created. Record descriptions in the Communication Section are processed in much the same way as those in the File Section. However, since they are not associated with files, no RCDTBL entries are made.

PHASE 12

Phase 12 (IKFCBL12) is loaded after phase 10 processing only if the source program contains a Report Section. Phase 12 reads the source statements of the Report Section of the Data Division, producing one complete Report Writer Subprogram (RWS) for each RD that it encounters. As it does so, it also:

- Scans its input for errors and generates any necessary E-text.
- Generates a listing of the Report Section on SYSPRINT if the SOURCE and NOLIB options are in effect or on SYSUT6 if LVL is in effect.
- Records information for later phases in TANER tables and in COMMON cells.

Phase 12 reads its input from SYSIN or from SYSUT4 if LIB or LSTCOMP is in effect. It writes its output, the RWS, in the form of Data IC-text on SYSUT3 and in P0-text on SYSUT2. Any E-text produced is also written on SYSUT2, intermingled with the P0-text. The input and output are summarized in Figure 7. The RWS is described, in "Appendix C. Report Writer Subprogram."

If the VERB option is in effect, Listing A-text is generated and passed to phase 60 or 64 so that the object program listing can include verb-names and procedure names. Each text element is simply a word in EBCDIC format preceded by a code and a count. For every Listing A-text element written, a card number element is written in P0-text. This card number (passed on through the changing text forms) indicates to phase 60 or 64 when to read a Listing A-text element.

OPERATIONS IN OTHER PHASES

In addition to normal processing of the Data IC- and P0-texts, other phases perform related operations in response to elements of the source program or of the RWS. These elements include the REPORT clause, the Report Section header, USE sentences, the Procedure Division verbs INITIATE, GENERATE, and TERMINATE, control-field save-area names, and REDEFINES clauses.

## REPORT CLAUSE

When a REPORT clause in an FD statement is encountered, routine TBLRPT of phase 10 primes the RWRPBL table (first REPORT clause only), sets a flag bit in the P1BTBL table (first REPORT clause only), and enters the report name into the RWRPBL table (each REPORT clause encountered). Phase 12 later checks the flag bit and, if it is not set, returns control to phase 00 without producing an RWS.

## REPORT SECTION HEADER

Upon encountering the Report Section Header, phase 10 sets the RPTWTR bit in the SWITCH cell in COMMON. Routine INT10 of phase 00 later checks the bit and calls phase 12 when it is set.

## USE SENTENCES

Upon encountering a USE sentence in the Declaratives Section of the Procedure Division, phase 1B:

- Generates REPORT-ORIGIN, a special Report Writer verb, to cause the address counter to be set to the first instruction in the RWS group routine resulting from the report group specified in the USE sentence.
- Inserts, at that point in the RWS routine, a link to the USE routine.
- Generates another Report Writer special verb, REPORT-REORIGIN, to reset the address counter.

Note: Report Writer verbs are discussed under "Elements of a Report Writer Subprogram" in "Appendix C. Report Writer Subprogram."

## PROCEDURE DIVISION VERBS

Upon encountering an INITIATE, GENERATE, or TERMINATE verb, phase 1B generates P0-text, and phase 51 later generates linkage between the main program and appropriate

routines in the RWS. The INITIATE verb results in a link to the INT-ROUT routine, TERMINATE to the LST-ROUT routine, GENERATE report-name to the 1ST-ROUT routine, and GENERATE detail-name to the DET-ROUT routine.

#### CONTROL-FIELD SAVE-AREA NAMES

Upon encountering control-field save-area names (which are generated by phase 12), phase 22 generates a dictionary entry consisting of the "-nnnn" name and the attributes of the control field which had been previously defined in the Data Division. Further discussion of control-field save-areas is provided under "Nonstandard Data-names" in "Appendix C: Report Writer Subprogram."

#### REDEFINES CLAUSE

Upon encountering Data IC-text for a Report Writer REDEFINES clause, phase 22 so processes it that the E-point name data item generated from the COLUMN clause points to the relative location in the print-line work area, RPT.LIN, equal to the integer specified in the COLUMN clause. When an item is later to be moved to RPT.LIN, the location can be determined from the E-point name. The length is taken from the PICTURE clause information in the dictionary attributes for the item. E-Point and RPT.LIN are discussed under "Nonstandard Data-names" in "Appendix C. Report Writer Subprogram."

#### PRODUCING THE REPORT WRITER SUBPROGRAM (RWS)

Generating a complete subprogram is the task of five routines in phase 12: RDSCAN, PROC01, PROC02, FLUSH, and GNSPRT. Routine GETDLM controls the flow of processing for phase 12. That flow is tailored to the particular source program, but the following discussion explains the general concept.

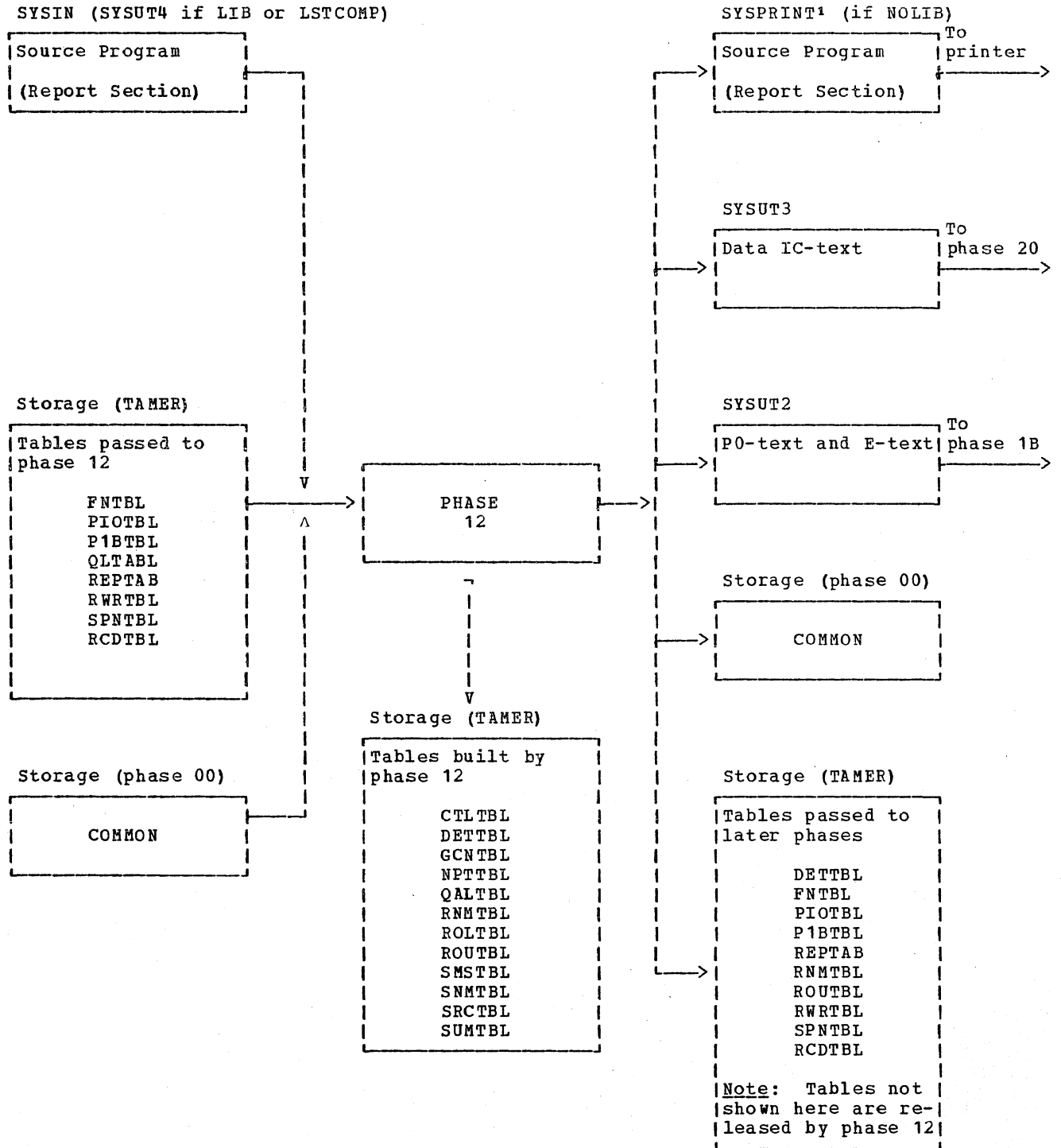
The RDSCAN routine processes the RD statement and is followed by routine PROC01, which processes the level-01 sentence. If that sentence is an elementary item, routine PROC02 is called upon to process each elementary-level clause until the entire sentence has been processed. At that point, the FLUSH routine is called to finish generating the group routine. If the sentence is the level-01 statement of a group item, routine PROC01 processes the sentence, and routine PROC02 is called to process the elementary and lower-level group items following. When that is done, routine FLUSH is called, does its processing, and the compiler goes on to the next level-01 level statement.

The PROC01-FLUSH or PROC01-PROC02-FLUSH loops continue until phase 12 has generated RWS group routines for all of the level-01 statements defined in the source program. Routine GNSPRT is then called on to complete the RWS by filling in the fixed and parametric routines and any necessary dummy group routines. Phase 12 then checks to see if the next logical record is an RD statement, in which case another RWS is needed and the process begins again with routine RDSCAN, or if it is the Procedure Division header, in which case phase 12, being finished, returns control to phase 00.

#### ROUTINE RDSCAN

The RDSCAN routine is responsible for processing the RD entry of the source program. After first ensuring that an RWS should be generated (by determining whether the RWRBTL table is primed), it reads each logical unit of the RD and processes it. The routine operates in a loop-type scan, checking each item to see if it is a period, CODE clause, CONTROL clause, or PAGE clause. If it is none of these, it is treated as an error.

Routine RDSCAN then sets appropriate switches and enters data into storage areas and tables. It then gets a new record and repeats the loop until it encounters a period. When this happens, control returns to routine GETDLM, which calls routine PROC01.



<sup>1</sup>SYSUT6 is used if the LVL option is in effect.

Figure 7. Phase 12 Input/Output Flow

#### ROUTINE PROC01

Routine PROC01 processes the level-01 record descriptions. Valid input for this routine includes the period and the NEXT GROUP, LINE, TYPE, and USAGE clauses. It operates in a loop-type scan and processes each clause in much the same way as the RDSCAN routine does. Since an level-01 elementary entry is permissible, other clauses can also be valid. Before assuming an error, routine PROC01, therefore, branches to the PROC02 routine to check for and process elementary-level clauses. Once a valid clause is processed in one or another of these routines, control returns to the beginning of the loop in routine PROC01.

Processing of the TYPE clause marks the generation of the initial coding for the group routines. Since the compiler has, at that point, enough input to begin the group routine, the first part of that routine is generated here.

#### ROUTINE PROC02

Routine PROC02 is entered when routine GETDLM encounters a level-02 through level-49 entry or, at entry point PRO2A, during PROC01's scan of an level-01 elementary item. Its operation is similar to that of routines RDSCAN and PROC01, except that checks are made so that control returns to routine PROC01 when appropriate.

#### ROUTINE FLUSH

When routine FLUSH is called, all the information needed to complete one group routine is available in the form of table entries, contents of data areas in storage, and switch settings. Routine FLUSH generates the exit coding for the group routine, and then returns control to routine GETDLM.

#### ROUTINE GNSPRT

Routine GNSPRT is called when the GETDLM routine encounters a new RD or the Procedure Division header. At this point, all group routines defined in the source program have been written on SYSUT2 (in P0-text), and all data needed to complete the RWS is in storage. Routine GNSPRT first writes out the necessary Data IC-text on SYSUT3 and then, in order:

1. Generates the WRT-ROUT routine.
2. Generates a dummy group routine for any of the following groups not defined in the source program: Control Heading Final, Control Footing Final, Page Footing, Page Heading, Report Heading, and Report Footing.
3. Generates the INT-ROUT routine.
4. Generates, if a PAGE LIMIT clause was specified in the source program, an ALS-ROUT routine and an RLS-ROUT routine. If no PAGE LIMIT clause was specified, the RWS contains neither of these two routines.
5. Generates one USM-ROUT routine for each TYPE IS DETAIL group specified under the RD statement being processed.
6. Generates in order, one each of the following routines: CTB-ROUT, RST-ROUT, 1ST-ROUT, LST-ROUT, and ROL-ROUT.
7. Generates any needed CTH-ROUT routines. A CTH-ROUT routine is needed for any control specified in the source program after the highest level (of FINAL) control. If the source program contains no TYPE IS CONTROL HEADING report description for such a control, routine GNSPRT generates a dummy group routine here to fill the need.
8. Generates any needed CTF-ROUT routines. A CTF-ROUT routine is needed under circumstances like those for a CTH-ROUT routine.
9. Generates one SAV-ROUT routine and one RET-ROUT routine.

#### GENERATING ERROR MESSAGES

Coincident with producing the RWS, phase 12 scans its input for syntax errors. Checks are made to ensure that each routine is both correct in itself and compatible with the rest of the RWS. If errors are detected, messages are written in E-text and recovery is attempted. When necessary, attempts to produce the particular RWS are abandoned. The E-text is written intermingled with P0-text, on SYSUT2.



GENERATING THE SOURCE LISTING

As each record is read from SYSIN, a check is made to determine if the SOURCE option is in effect. If so, the source statement is copied out on SYSPRINT (or SYSUT6 for LVL option).

phases to use. For example, phase 12 builds the ROUTBL table, which contains the specific GN numbers assigned to certain RWS routines. Phase 1B needs this information to process INITIATE, TERMINATE, GENERATE, and USE BEFORE REPORTING statements. Such items are stored in TAMEX tables and in cells in COMMON. For more details on this subject, see Figure 7 and "Section 5. Data Areas."

INFORMATION FOR LATER PHASES

During its processing, phase 12 stores various types of information for later

## PHASE 1B

Phase 1B (IKFCBL1B) reads the Procedure Division of the source program. It is entered, via phase 00, when the GETDLM routine in phase 10 or 12 encounters the Procedure Division header. As it reads each card image in the Procedure Division, it performs the following major functions:

- Encoding the Procedure Division into Procedure IC-text (P0-text format).
- Creating dictionary entries for procedure-names.
- Writing the Procedure Division on SYSPRINT (SYSUT6 for LVL option) if the SOURCE option was specified by the user.
- Testing the BATCHSW switch in COMMON for a hexadecimal '40' (ENDFOUND) value, if the BATCH option was specified by the user. If this value is on, the last card in the current compilation has been read in phase 10 and the switch is set off. Phase 1B then returns to phase 00. Otherwise, phase 1B checks for the CBL card, which is a source delimiter. When a CBL card is found, phase 1B sets on the hexadecimal '08' (CARDHELD) value in the BATCHSW switch in COMMON, stores the address of the CBL card in COMMON location ADDR CARD, and returns to phase 00.

Phase 1B routines first process the out-of-line procedures contained in the Declaratives Section. (This processing is described under "Declaratives" below.) Then the in-line program is processed.

Tables passed from phase 10 and used by phase 1B include the P1BTBL, PIOTBL, FNTBL, RCDTBL, SPNTBL, and TOTBTBL tables. Tables passed from phase 12 for Report Writer operations are the RWRBTBL, DETTBL, ROUTBL, and RNMTBL tables. The PNTABL and PNQTBL tables are built during phase 1B. If the VBREF or VBSUM option is in effect, phase 1B will create the VERBDEF Tamer table. (See "Section 5. Data Areas" for formats of all these tables). Dictionary entries are made for all source procedure-names. Procedure IC-text (in P0-text format) is generated from the procedure statements. Formats for these texts and the dictionary are given in "Section 5. Data Areas."

The PDSCN routine controls Procedure Division processing. The GETCRD and GETWD routines supply the input in a manner similar to the description under "Major Working Routines" in the "Phase 10" chapter. These and all other routines used by both phases are indicated by an asterisk in the subroutine directories for both phases. Such routines do not remain in storage from phase 10, but are reloaded with phase 1B.

A major activity of phase 1B is writing P0-text. This text is, roughly, the source program Procedure Division recoded into a form acceptable to later phases. Logical units (source program words) are processed, encoded, and written out one at a time. Some information, such as card numbers, is generated for P0-text. All user-assigned names are passed unchanged (preceded by code and count fields) from the source text. Verbs and other COBOL words are replaced by unique 2-byte codes. For the complete text formats, see "Section 5. Data Areas."

If the VERB option is in effect, Listing A-text is generated and passed to phase 60 or 64 so that the object program listing can include verb-names and procedure names. Each text element is simply a word in EBCDIC format preceded by a code and a count. For every Listing A-text element written, a card number element is written in P0-text. This card number (passed on through the changing text forms) indicates to phase 60 or 64 when to read a Listing A-text element.

### PROCEDURE-NAMES

At its point of definition, each procedure-name (paragraph-name or section-name) is given a PN number. The point of definition is that point at which the name appears in Area A of the source program. PN numbers are assigned sequentially starting with 1 from cell PNCTR in COMMON. The procedure-name is entered in the dictionary, and written in P0-text.

ENTERING PROCEDURE-NAMES IN THE DICTIONARY

The dictionary is a repository for all information that can be gathered about each user-assigned name in the source program and dummy procedure-names for the beginning and end of the Procedure Division and of the Declaratives Section. Such information about a name comprises its attributes. After the dictionary has been completed by phase 21, the attributes are substituted for the name itself in the Procedure IC-text produced by phase 3. In later Procedure IC-text processing by phases 4, 45, 50, and 51, all information about the name appears conveniently in the text stream, and the space required by the dictionary can be released.

Dictionary entries for procedure-names are made, using the DICOT and HASH tables, by phase 1B. A procedure-name may occur in the source program at its point of definition, where it is called a left-hand name. The name may also occur as an operand in a statement such as "GO TO procedure-name." A procedure-name used as an operand is called a right-hand name.

A dictionary entry is made for each left-hand name as it occurs in the source text, and some of its attribute bits are set at this time. Other attribute information is not known until all occurrences of the name as a right-hand name have been read. (Verbs associated with right-hand names are discussed under "Declaratives" and "Procedure-Branching Verbs" later in this chapter.)

The phase 1B routines do not make entries directly in the dictionary by themselves. Rather, they call a special group of routines called ACCESS routines, which are resident during phase 1B in the area of storage below the phase 1B code. These routines are designed especially to build and use the dictionary. They are also resident during phases 22, 21, 25, and 3, the other phases which use the dictionary. They are described in "Appendix A: Table and Dictionary Handling."

USING THE PNTABL AND PNQTBL TABLES

The PNTABL and PNQTBL tables are used to store certain attributes of procedure-names before these attributes are entered in the dictionary. The PNQTBL table contains entries for certain procedure-names that are qualified by section-names, while the PNTABL table contains entries for certain nonqualified procedure-names.

Storing Information

Entries are made in the PNTABL or the PNQTBL tables when certain verbs are encountered (see "Procedure-Branching Verbs" and "DEBUG" below), or when a declarative section is processed (see "Declaratives" below). Entries are also made for dummy names. Information may be added to a table entry if the procedure-name occurs again before the dictionary entry is found. A new entry will be made for a previously entered procedure-name only if the old entry has already been deleted.

The formats of these entries correspond exactly to the format of dictionary attributes for procedure-names. Note that a source statement may refer to a procedure-name which has not yet been defined and, therefore, is not yet in the dictionary.

Moving Information into the Dictionary

The dictionary is searched at the end of every source program section. When the procedure-name in the table matches the procedure-name in the dictionary, information from the switches in the table is recorded in the dictionary attributes field, and the table entry is deleted. The dictionary search techniques are different for the PNTABL and PNQTBL tables.

At the end of the first source program section, the dictionary is searched for every procedure-name in the PNTABL. If a dictionary entry is found, the information is transferred and the PNTABL entry is deleted. If no entry yet exists, the search bit is turned on in the PNTABL entry. This bit indicates that the procedure-name does not appear in the portion of the dictionary previously searched. At the end of the next source program section, another search is performed for each procedure-name then in the PNTABL table. Whenever the search bit is on for a given entry, the search is restricted to the dictionary entries of the most recent source program section (see "LATACP" in "Appendix A. Table and Dictionary Handling").

The PNQTBL table contains the source program section-name qualifiers of the procedure-names entered. Thus, if the source program section named in the table has already been processed, a corresponding entry can be found in the dictionary by a limited search (see "LDELNM" and "LATACP" in "Appendix A. Table and Dictionary

Handling"). The table entry is deleted after the information has been transferred to the dictionary entry. If the section-name has not yet been encountered, no search is made. Therefore, this table does not require search bits.

Figure 8 shows in diagram form an example of the dictionary search. It explains how the search would proceed for several hypothetical procedure-names. Note that the figure does not represent table formats.

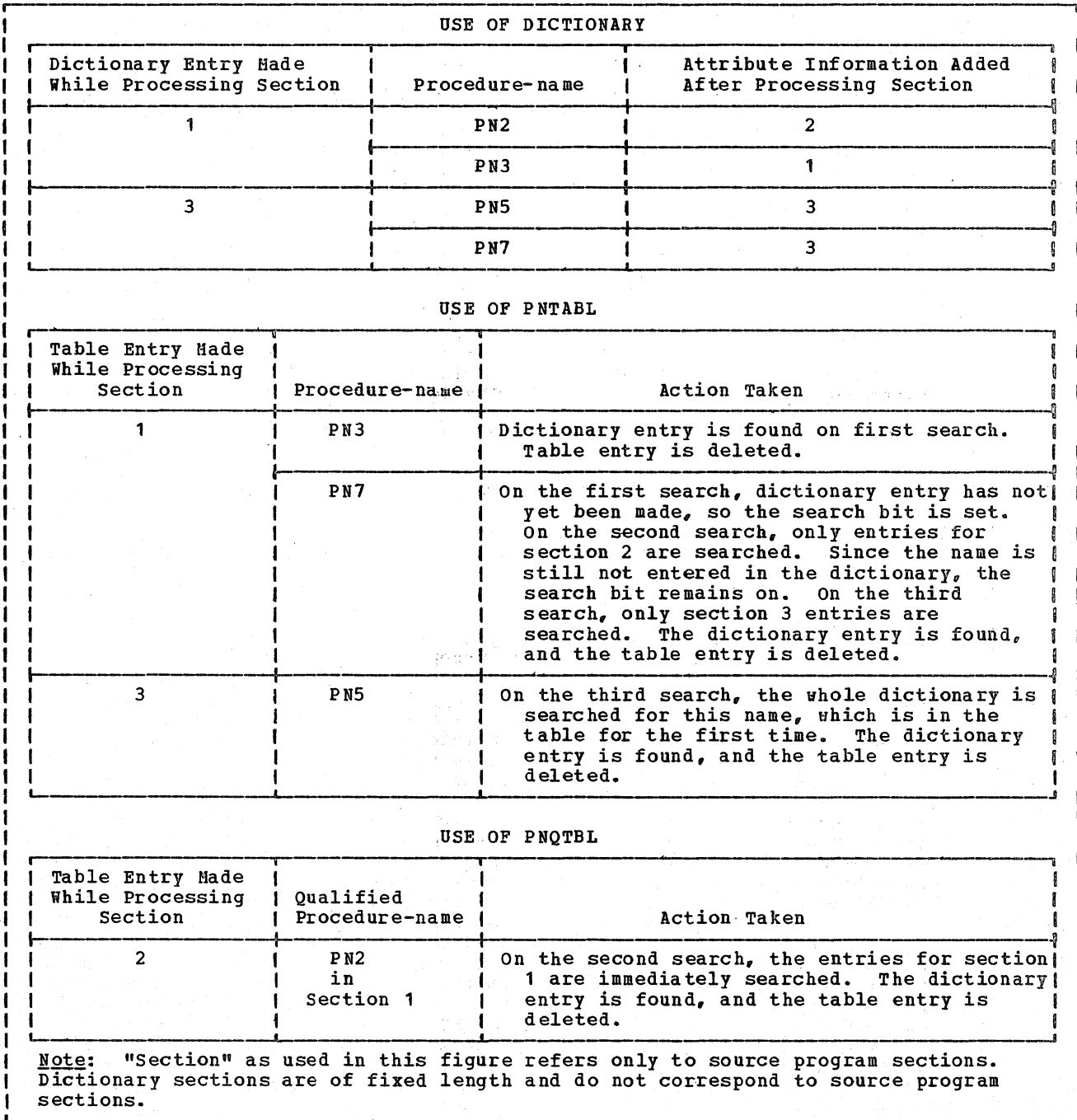


Figure 8. Entering PNTABL and PNQTBL Information in the Dictionary

**PRIORITY CHECKING FOR SEGMENTATION**

For each section-name, the segmentation priority is entered in the dictionary. If no priority number was specified, zero is entered as the priority. (In a nonsegmented program, all sections are given a zero priority.) If a priority number was specified, its value is compared to the value of the SEGLMT cell in COMMON (set by phase 10; see "SEGMENT-LIMIT Clause" in the "Phase 10" chapter). If the priority number of the section-name is less than SEGLMT, it indicates that the section is part of the root segment. In this case, the priority number of the section-name is entered in the dictionary as zero. If the priority number exceeds SEGLMT, it is entered as specified.

Each time a section-name is found whose priority exceeds the value of SEGLMT, a switch called SEGSW, whose location is internal to phase 1B, is turned on. If, at the end of Procedure Division processing, this switch still contains zero, it means that the program is not segmented, and SEGLMT is set to hexadecimal 'FF'. If the switch is on, SEGLMT is left as it is. The value of SEGLMT is used by later phases to determine whether the program is segmented.

**VERBS**

All verbs are encoded and written into P0-text. In addition, the verbs discussed below require special handling.

**PROCEDURE-BRANCHING VERBS**

Procedure-branching verbs use the PNTABL and PNQTL tables. Entries in these tables are used to set some of the attribute bits in the dictionary entries for the procedure-names. The process by which dictionary entries are made from these tables is described in this chapter under "Using the PNTABL and PNQTL Tables."

**GO TO:** The left-hand name (the procedure-name appearing in Area A, not the object of the GO TO statement) is entered into the table. (A GO TO statement preceded by a procedure-name definition may be the object of an ALTER statement elsewhere in the program.) If the DEPENDING ON option is used, no entry is made.

**EXIT:** The left-hand name is entered.

**ALTER:** The procedure-name following the word ALTER and the procedure-name following the phrase TO PROCEED TO are entered.

**PERFORM...THRU:** The procedure-name following the THRU is entered. If the THRU option is not used, the procedure-name following PERFORM is entered. (This entry directs phase 4 to provide a return of control at the end of the PERFORM procedure.)

**INPUT/OUTPUT VERBS**

Switches are set in the PIOTBL entry for the file named in the input/output statement. These switches indicate to phase 21 how the file is used. In addition, the following processing takes place:

**DELETE:** The FILSVB subroutine checks that the file name specified in the DELETE verb was previously specified in a SELECT statement. The subroutine turns on the appropriate bit in the PIOTBL table if the filename is valid.

**OPEN:** If label- or error-processing declaratives were written for this file, GNs (generated procedure-name references) for the declaratives are encoded in the P0-text following the file-names in the OPEN statements. The handling of these declaratives is described below under "Declaratives."

**READ:** The PIOTBL entry for the file is checked to determine whether the RERUN bit is on. If it is, a special verb code is used. The REDSVB routine also checks for the word NEXT in the READ verb. If present, the routine turns on the appropriate bit in the PIOTBL table.

**WRITE, REWRITE:** The record-name is sought in the RCDTBL table and from its FNTBL pointer the file-name is found. The file-name is then included in the P0-text entry (in the form WRITE file-name record-name). For a WRITE statement, the PIOTBL entry for the file is checked to determine whether the RERUN bit is on. If it is, a special verb code is used in the P0-text. If the ADVANCING mnemonic-name option of the WRITE statement is used, the mnemonic-name is sought in the SPNTBL table and replaced with the proper special-name. If an INVALID KEY option is used, phase 1B sets the appropriate PIOTBL bit to 1.

**START:** The STTSVB subroutine checks that the file name specified in the START verb was previously specified in a SELECT statement. The subroutine turns on the appropriate bit in the PIOTBL table if the filename is valid. The subroutine also checks for a KEY of REFERENCE and/or an INVALID KEY clause. If either or both are found, the subroutine turns on the appropriate bit(s) in the PIOTBL table.

## OTHER VERBS

**ACCEPT, DISPLAY:** The mnemonic-name used is sought in the SPNTBL table and replaced by the proper special-name.

**DEBUG:** The attribute bits in the dictionary entry for the specified paragraph are set, using the PNTABL and PNQTLB tables as described above.

**EXHIBIT:** A special EXHIBIT data-name is generated.

**READY:** The SWTRCE switch in the SWITCH cell in COMMON is set.

**SORT/MERGE:** If the USING or GIVING options are specified, the appropriate PIOTBL bits are set in the entries for the files named. If these files are also specified in label- or error-processing declaratives, a special code is used for the USING or GIVING elements in the P0-text.

If the COLLATING SEQUENCE phase is specified, the ALPHTBL is scanned to ensure that the phase is valid.

**Report Writer Verbs:** See the chapter "Report Writer Subprogram."

## DECLARATIVES

When a declarative section is encountered, the section-name (and paragraph-names, if any) are entered into the dictionary. A PNTABL entry is made for the section name, with the declarative bit and the appropriate bit to identify the type of declarative set. Every paragraph-name in the section is also entered in this table, with only the declarative bit set. These bits are used later to set dictionary attribute bits.

Each label or error declarative is given a GN (generated procedure-name) number. These numbers are assigned sequentially starting with 1 from the GNCTR cell of COMMON. If the USE sentence specified file-name, the GNS are entered in the appropriate fields of the FNTBL entry for the file. If the USE sentence specified INPUT, OUTPUT, EXTEND, or I-O, the GN number is entered in a work area. If a USE FOR ERROR declarative statement is encountered, the appropriate PIOTBL table bit is set to 1.

These work areas and tables are checked when OPEN verbs are encountered. For each file-name specified in an OPEN statement, the corresponding FNTBL entry is inspected.

If GNS were entered in the FNTBL table during declarative processing, they are inserted in the P0-text. Otherwise, the work areas for the particular OPEN option are used for GNS. If an ON INPUT GN for an error declarative is found in the work area, the GN number is inserted in all OPEN INPUT P0-text entries. The GNS for label declaratives are determined for files whose FD statements include a LABEL RECORD IS data-name clause. OUTPUT and I-O GNS are handled the same way.

The USESVB routine builds the GVNMTBL table containing the fully qualified data-names used in the GIVING option of the STANDARD ERROR/EXCEPTION PROCEDURE declarative for VSAM files. The routine also builds the GVFNTBL table when a VSAM file has been specified in the ON option of the declarative.

For a USE FOR DEBUGGING declarative, the USEGEN label is used for the DEBUG processing code, and will be entered only from DCLSCN at the start of the declaratives. Following USEMSG, a branch will be inserted to label USEGN1. At label USEGN1 a check is made for FOR followed by DEBUGGING; if found, the USEDBG code is executed; otherwise the USE verb code (USECON) is restored to CURCOD and a branch is taken to continue processing at USEGN1.

The USEDBG verb is generated if a USE FOR DEBUGGING declarative has at least one valid operand. USEDBG checks the V2BUGON switch. If V2BUGON is not on, USEDBG ignores all code for this USE sections and scans to the next section-name, USE verb, or END DECLARATIVES. If V2BUGON is on, USEDBG checks for ALL PROCEDURES; if found, BGALLPRC is tested to see if it has been specified before. If so, a message is produced indicating that the second occurrence of ALL PROCEDURES is ignored. If valid, a procedure-name number for the section-name, prior to USE, is entered in BGALLPN and its priority number is entered in BGALLPRI, and a USEDBG verb is generated. V2BUGDCL is set on to indicate a valid USE FOR DEBUGGING declarative has been found. Control is transferred to USEDCB which checks the BCD-name operand. USEBCD generates an alphanumeric literal representing the BCD-name operand. If it is a qualified-name, it goes to QLTABL to generate an alphanumeric literal for up to 30 bytes containing the name and its qualifiers, separated by OF. If it is a BCD-name, the contents of CURBCD can be used. Then the name is generated and end-of-sentence is checked for; if so, exit is to VRBENT, if not, return to search for more operands. If no operands are found for the USE verb, a message is produced and the USE sentence is discarded.

PHASE 20

Phase 20 (IKFCBL20) is the third of five phases that process the Data Division. It follows phase 1B, overlaying it in storage. Phase 20 is called by routine LINKPH1 of phase 00 and returns control to phase 00 when it encounters an end-of-file condition on SYSUT3. The primary concerns of phase 20 are the VALUE and PICTURE clauses of the data descriptions, which it translates from Data IC-text into ATF-text.

Phase 20 processing is initiated and controlled by the BEGIN routine. It reads each Data IC-text element and, from each LD entry, it computes and writes a partial dictionary entry to be passed to phase 22.

After completing the partial entry, phase 20 writes it on SYSUT4 (the format of the entry is called ATF-text) and reads the next Data IC-text element, continuing until SYSUT3 has been exhausted. All Data IC-text for FDS, SDS, CDS, and keys for table handling and any E-text encountered is copied unchanged onto SYSUT4.

Phase 20 also scans its input for syntax compatibility and error conditions, producing any necessary E-text; it produces and passes two tables, the VALTRU and the VALGRP, to later phases. The input and output for this phase are summarized in Figure 9.

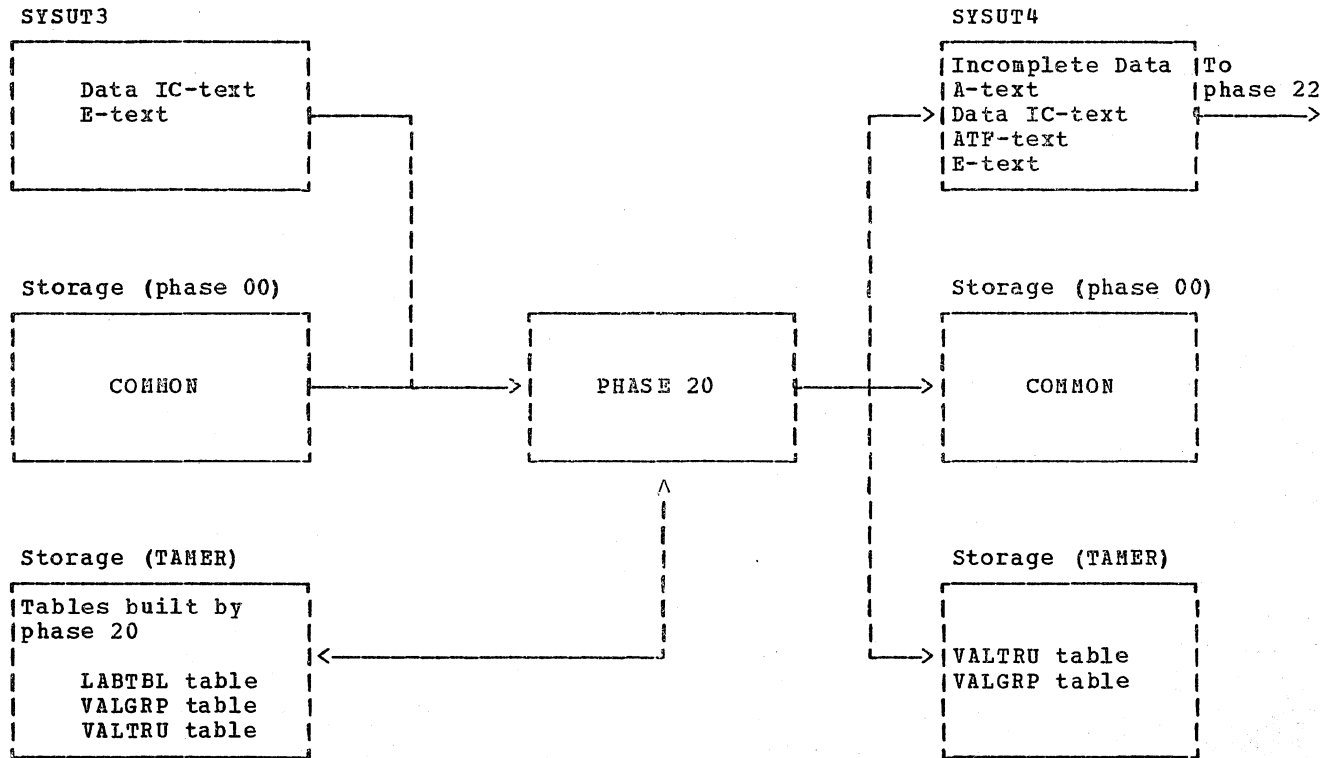


Figure 9. Phase 20 Input/Output Flow

### TRANSLATING LD ENTRIES INTO ATF-TEXT

Routine BEGIN first determines whether the element read is an LD element; that is, one resulting from a source program record description entry of level numbers 01 through 49, 66, 77, or 88. If so, BEGIN stores the current input card number in COMMON and calls on routine LDTEXT. LDTEXT copies the element from the input buffer into a work area, called ATFTXT, reads the next record (Data IC-text element) into the buffer, compares the current level number to the next element's level number to determine whether it is a group or an elementary item, and calls the appropriate routines to create the ATF-text.

### PROCESSING ELEMENTARY ITEMS

For an elementary item, the LDTEXT routines described below produce a portion of an ATF-text element that contains fields identical to a dictionary entry except for the addressing parameters. Routine DICTBD of phase 22 fills these in later.

If there was a PICTURE flag in the Data IC-text for the elementary item, the LDTEXT routine calls routine GSPICT to distribute the PICTURE into work areas. The kind of character is stored in work area IPT and the number of occurrences of that character in IPLT. Subroutines, depending on the type of the PICTURE, determine the length of the item and its attributes. The attributes are entered in the variable information field of the ATF-text element.

An indication of how many subscripts are needed to refer to the item is set in the text element by subroutine BMBSRN. The REDEFINES bit is set from the REDEFINES flag in the Data IC-text, and the object of the REDEFINES clause is saved for processing by phase 22. (If the REDEFINES clause is internal, that is, generated for the Report Section, and the subject of the clause is a name plus a displacement, phase 22 adjusts the addressing parameters of the object of the clause to reflect the displacement.) The major code, which is changed only when a new section header is encountered, is moved from a work area to an ATF-text field. In addition, the level numbers are normalized as an aid to phase 3.

Routine BUSAGE utilizes the USAGE information in the Data IC-text to determine the size of the elementary item if there was no PICTURE. It provides enough information to set the minor code field in the ATF-text element to the type

of the entry and to fill in the variable information field with a description of the item. If there was a PICTURE, the USAGE information, together with the PICTURE, provides enough information to set the minor code and variable information fields.

If a RENAMES clause is associated with a data item, no partial dictionary entry exists in the ATF-text element, and all processing is done by phase 22. The BCD names are passed on unchanged from the Data IC-text element.

### PROCESSING GROUP ITEMS

For a group item, the LDTEXT routines produce a portion of an ATF-text element that is identical to a dictionary entry except for the addressing parameters and the length of the group. These are later filled in by phase 22.

The processing is the same as that for elementary items with two exceptions. Routine BUSAGE saves the USAGE, in area GUI, to verify the USAGE of the elementary items. Routine SRCHTB passes the keys for table handling, if any, unchanged to phase 22.

### PRODUCING INCOMPLETE DATA A-TEXT

Phase 20 generates incomplete Data A-text elements for constants defined by VALUE clauses. Information for constructing this text comes from the Data IC-text read from SYSUT3. For LD entries with VALUE clauses, the value is given in the Data IC-text element and is entered directly by routine VALGEN into the incomplete Data A-text element. Constants defined by VALUE IS SERIES clauses are discussed under "Building Tables for Later Phases" in this chapter. For the formats of Data A-text and Data IC-text, see "Section 5. Data Areas."

### PROCESSING FILE SECTION ENTRIES

When it encounters the File Section header, the BEGIN routine transfers control to routine FILEST, which controls the processing of the section. Routine FILEST uses the BSUBRN routine to read the Data IC-text elements. If the next item read is a critical program break or if EOF is encountered, routine FILEST returns control to routine BEGIN.



PROCESSING COMMUNICATION SECTION ENTRIES

When it encounters the Communication Section header, the BEGIN routine transfers control to routine COMSCT, which controls the processing of the section. CD entries are passed unchanged by routine CDTEXT; LD entries are processed by routine LDTEXT as described earlier.

PROCESSING ERRORS

As phase 20 processes Data IC-text, the clauses are checked to determine whether

they are allowed to be used together. The following is an example of the checking that is performed.

When routine LDTEXT processes Data IC-text elements for LD entries, some of the clauses are processed before a determination is made of whether the item is a group or elementary item. Then, when the LDTEXT routine determines whether the item is a group or elementary item, it eliminates any invalid clauses. For example, if a PICTURE clause is given for a group item, routine LDTEXT processes it. Then when it determines the item is a group item, routine ERRTN issues E-text for the invalid PICTURE clause.

## PHASE 22

Phase 22 (IKFCBL22) is the fourth of five phases that process the Data Division. Phase 22 follows phase 20, overlaying it in storage. Its major functions are producing dictionary entries, completing Data-A text, and generating Q-routines.

Phase 22 processing is initiated and controlled by the DIRECTOR routine. A Data IC-text or ATF-text element is read from SYSUT4 and distributed to work areas. Routine DICTBD then completes fields in the entry and places it in the dictionary. While building the dictionary entry, phase 22 also checks for syntax compatibility and error conditions. After phase 22 has completed the dictionary entry for a given text element, it picks up the next element for processing, continuing until SYSUT4 has been exhausted. All Data IC-text for FDS and SDs and any E-text encountered is copied unchanged onto SYSUT3.

Incomplete index-name entries (prefix 04) are entered into the dictionary by routine READF4 when they are encountered. Later, information is filled in by routine XTEN, a subroutine of DICTBD.

The input and output for this phase are summarized in Figure 10.

### BUILDING DICTIONARY ENTRIES

The dictionary is used to store information about procedure-names and data operands and is the product of phases 1B, 21, and 22. Phase 22 stores the current card number,

generated during phase 10 for each input card in COMMON. It then calls the appropriate routine for preprocessing of the data item, and after the preprocessing is finished, it calls routine DICTBD to complete the entries. The routine makes either complete or dummy dictionary entries.

### DICTIONARY PREPROCESSING

RENAMES Entries: If a RENAMES clause is associated with a data item, routine RENAMS goes to the dictionary and locates the data item or items being renamed. The routine picks up the attributes and addressing parameters for the dictionary entry or entries and assigns them to the RENAMES item. The routine then places the completed entry for the RENAMES item into the dictionary. The entire dictionary entry for a RENAMES item is formulated by the RENAMS routine.

LD Entries: Before the dictionary build routine is called to complete the dictionary entry for an elementary item, routine LDTEXT obtains a dictionary pointer for the item by calling an ACCESS routine, GETPTR. (ACCESS routines are dictionary-handling routines.)

A delimiter pointer is needed for group items.

Level-88 entries are put into the dictionary directly by the input routine READF4.

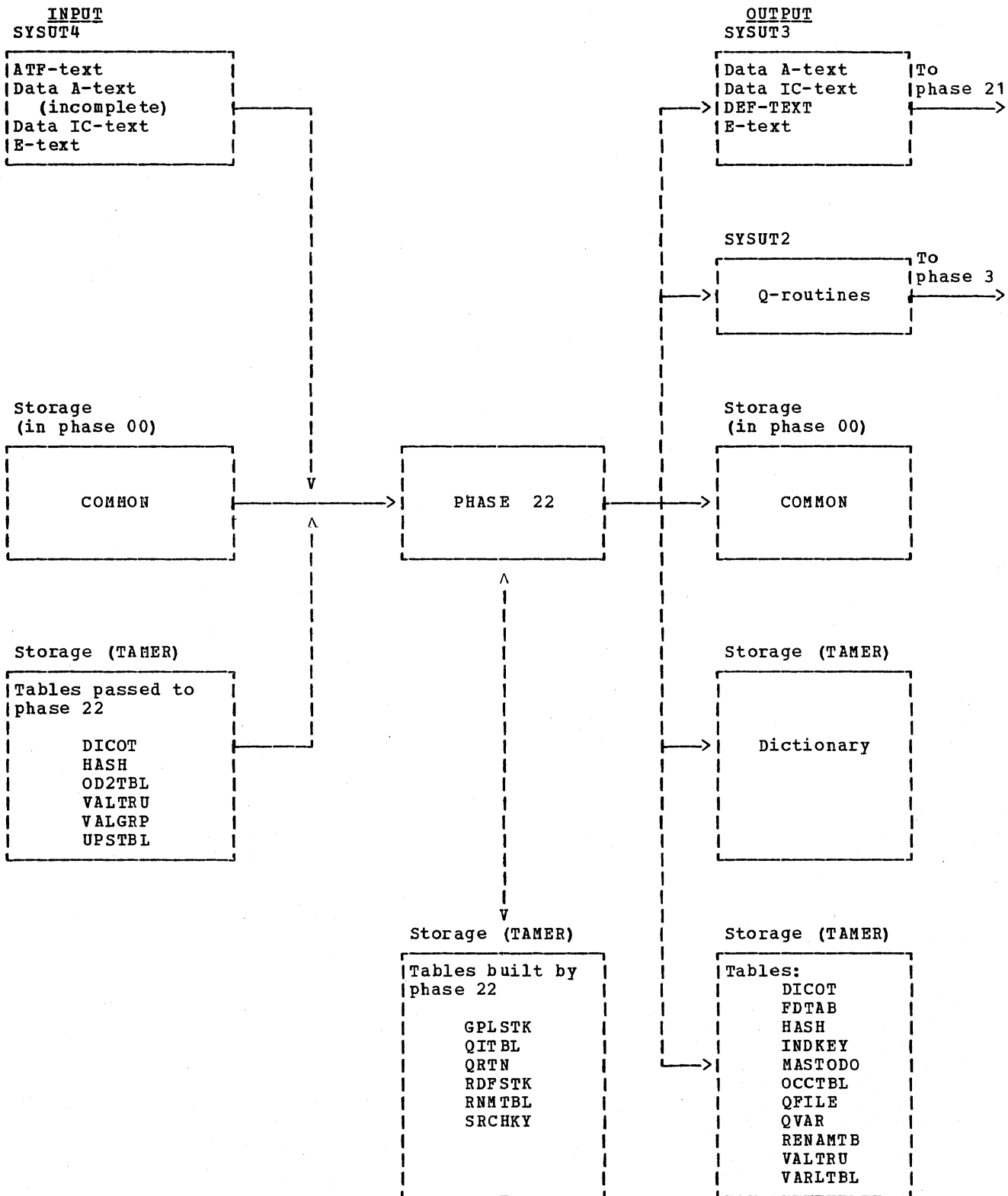


Figure 10. Phase 22 Input/Output Flow

FD Dictionary Entries: A skeleton dictionary entry is created by routine FSTXT. This entry contains only the file name; the attributes are filled in by phase 21. The length of the VSAM FD dictionary attributes is determined by phase 22 when making the skeleton entry. This is true for all other access methods. Routine FSTXT writes the Data IC-text for the FD on SYSUT3. Phase 22 determines if there is an ISAM file which has no RESERVE NO clause and is opened INPUT or I-O.

Since Phase 21 processes the Data IC-text for FDs, routine FSTXT passes this text to SYSUT3 (the same file on which phase 22 writes Data A-text) except for user label record information.

A dictionary pointer is obtained and processing of the entry is completed by routine DICTBD and its subroutine FST000.

A skeleton dictionary entry is also created for any LINAGE-COUNTERS specified for an FD.

CD Dictionary Entries: Routine CDTEXT does the processing for the CD dictionary entry; it creates a dummy level-01 entry and dummy level-02 entries for each CD-name from predefined attributes. Routine DICTBD enters these in the dictionary.

Partial RD Dictionary Entries: Routine RDTXT does most of the processing for the RD dictionary entry. At the end of this processing, the entry is complete and entered in the dictionary by routine DICTBD.

SD Dictionary Entries: SD entries are handled like FD entries. Routine SDTXT performs this processing.

#### COMPLETING DICTIONARY ENTRIES

The dictionary build routine, DICTBD, completes these dictionary entries which were begun in phase 20 by filling in addressing information. Each data item is addressed by a base locator number and a displacement.

The addressing parameter field has three parts, called i, d, and k, where i specifies the type of base locator (BL, BLL, or SBL), d specifies the displacement of the item from the beginning of the area controlled by the base locator, and k specifies the base locator number.

A base locator number is assigned to the beginning of each major data area, such as the Working-Storage Section, and to each FD

and SD entry. Then the displacement of each item in these areas from the beginning of the area is calculated. If the items in the area occupy more than 4,096 bytes, a second base locator number is assigned to the second 4,096 bytes, etc. (In this case, CD and RD entries are considered to be an extension of the Working-Storage Section and the same base locators are used.)

There are three types of base locators (BL, BLL, and SBL) depending on the type of data area. Base locator numbers are assigned sequentially from counters in the COMMON area.

Type	Counter	Use
BL	BLCTR	BL numbers are assigned to the Working-Storage Section, the Communication Section, the Report Section, and to each FD, CD, and RD entry.
BLL	BLLCTR	BLL numbers are assigned to the Linkage Section, the DEBUG-ITEM special register, if specified, label records, and to each SD entry.
SBL	SBLCTR	See "Q-routine Generation" in this chapter.

Completing Working-Storage Section Entries: The Working-Storage Section contains only LD entries. Routine DICTBD completes the LD dictionary entries by filling in the addressing parameter field for group and elementary items and by determining the length and the delimiter pointer for group items.

Routine DICTBD assigns a base locator number to the beginning of the Working-Storage Section. The type of the base locator number is BL, and the base locator number is the next available number from field BLCTR in COMMON. The d part of the addressing parameter is obtained from the LOCCTR counter in COMMON. Each time a data item is processed; the counter is incremented by the number of bytes the element will occupy at object time.

Routine DICTBD uses the GPLSTK table to keep track of the length of group items. It enters the length and the delimiter pointer (the dictionary pointer of the group delimiter) in the dictionary entry for the group. It also deletes the GPLSTK table entry for the group.

Note: The lengths of level-77 items are not added to the GPLSTK table since they are independent items.

If an item contains a REDEFINES clause, routine DICTBD calls routines REDEF and RDSYN to process the item using tables RDFSTK and RNMTBL. The REDEF routine makes an entry in the RDFSTK table, giving the length of the REDEFINES object and the level number and current addressing parameters of the REDEFINES subject. Then the REDEF routine assigns the addressing parameters of the REDEFINES object to the REDEFINES subject. An entry is also made in the RNMTBL table, giving the level number, dictionary pointer, and length of the object of the REDEFINES.

When an item is encountered with a level number less than or equal to the last level number in the RDFSTK table, it is assigned the addressing parameters from the entry.

The length of the REDEFINES object is saved in the RDFSTK table to see whether the length of the REDEFINES subject is less than or equal to it. If the REDEFINES subject is a group item, its length is determined by the GPLSTK table. If it is an elementary item, routine RDSYN determines its length. (Routine RDSYN also checks the name of the REDEFINES object against the entries in the RNMTBL table to see whether or not it is valid.)

Table RNMTBL is also used if there are a series of items with REDEFINES clauses. It saves the names so that an item can redefine an item that does not immediately precede it in the series.

Completing File Section Entries: Routine DICTBD uses its subroutine FST000 to complete dictionary entries for FDs. Subroutine FST000 performs two major functions:

- It resolves the previous FD, if any.
- It completes the processing of the current FD, if any.

Routine DICTBD processes SD entries in the same way it processes FD entries, except that the type of base locator number assigned is BLL. These are obtained from field BLLCTR in COMMON, and the first BLL number used is 3.

Completing Linkage Section Entries: Linkage Section entries are processed the same as Working-Storage Section entries, except that the type of base locator number assigned is BLL. For a label record item, the first BLL is assigned. For other items in the Linkage Section, BLL numbers are assigned starting with the first BLL number not used for an SD entry. All level-77 items and all group items starting with level-01 in the Linkage Section are assigned unique BLL numbers.

Completing Communication Section Entries: Routine CDTEXT calls routine DICTBD to process the dummy CD entries that CDTEXT has created. Location counter values are saved so that every level-01 entry under a CD starts at the same location. Base locator numbers for level-01 entries are assigned to these items as though they were in the Working-Storage Section. The location counter value for the CD FOR INITIAL INPUT, if specified, is saved in CDLCCTR in COMMON.

Completing Report Section Entries: Routine DICTBD adds no information to Report Section entries before it puts them in the dictionary.

#### GENERATING DATA A-TEXT

Phase 22 completes the incomplete Data A-text elements passed to it by phase 20 by adding the location counter values. The two prefix bytes (the X'10' indicator and the length count affixed by phase 20) are left to serve as an indicator to phase 21 that the text element needs no further processing. Phase 21 deletes the first 2 bytes and then passes it unchanged to phase 6 or 64 and selects the Data IC-text elements (for FDs and SDs) for translation into Data A-text.

Phase 22 generates five types of Data A-text elements itself. While doing so, it prefixes them with the same two bytes of information discussed above. The four types are:

- Working-Storage Section address elements.
- Constants from VALUE clauses.
- Data-name DEF elements.
- Verb DEF elements.
- Q-routine identification elements.

To create these elements, phase 22 uses information stored in COMMON, tables GPLSTK, VERBDEF, and VALGRP, Data IC-text created in phase 10 or 12 and passed by phase 20, and ATF-text created by phase 20.

#### Q-ROUTINE GENERATION

Phase 22 uses the following tables to generate Q-routines: OD2TBL, QFILE, QVAR, OBJSUB, QITBL, and QRTN. The OD2TBL table is created by phase 10 and the other tables

by phase 22. The QFILE table is passed to phases 21 and 3 and the QVAR table is passed to phase 3; the OD2TBL, OBJSUB, QITBL, and QRTN tables are released by phase 22. (When the SYMDMP option is in effect, however, the QITBL and QRTN tables are passed to phase 25.) the OD2TBL contains the qualified names of objects of OCCURS...DEPENDING ON clauses.

Routine QVARBD combines the information contained in the OD2TBL, the QRTN, the OBJSUB, and the QITBL and QFILE tables into the QVAR and QFILE tables for phase 3; the QFILE table may be updated by phase 21. The routine then releases the OD2TBL, QRTN, and QITBL tables. (When SYMDMP is in effect, it releases only the OD2TBL table.)

If phase 10 created an OD2TBL table, phase 22 checks each elementary item that it processes to see whether or not it is in the OD2TBL table. If it is, phase 22 sets the dictionary entries of the item and all its groups to reflect that they are objects of OCCURS...DEPENDING ON clauses. If it is a group item, routine XTEN performs the processing; if it is an elementary item, routine ELIPR handles the processing. Routine ELIODO then places the dictionary pointer for the item and a pointer to the related OD2TBL entry in the QITBL table. If an object of an OCCURS...DEPENDING ON clause is encountered while processing the File Section, its OD2TBL table displacement is placed in the OBJSUB table.

When phase 22 encounters an ATF-text element for an LD entry with a pointer to the OD2TBL table (that is, the item was described with an OCCURS...DEPENDING ON clause), routine INTVLC marks all the group items currently in the GPLSTK table as variable in length by assigning a VLC (variable-length cell) number from field VLLCTR in COMMON to each item. If a subject of an OCCURS...DEPENDING ON clause is encountered while processing the File Section, its GN number is placed in the OBJSUB table.

In addition, if an item follows a variable-length field and is not a new file or record, it is variably located. To each of these items, phase 22 assigns an SBL (secondary base locator) number from field SBLCTR in COMMON. At execution time, there are secondary base locator cells (one for each SBL number) in the Task Global Table that contain the current location of the variably located field. Phase 22 generates Q-routines to calculate initial values and changes in these secondary base locator cells.

Whenever phase 22 generates Q-Routine text, a determination is made to see whether or not it is the first time that

Q-Routine text has been generated for this record. If it is the first time, a GN number is generated and routine QBUILD places it in front of the Q-Routine text for identification. This routine then makes an entry in the QRTN table containing the GN and the pointer to the OD2TBL table. If it is not the first time, the QRTN table is checked to see whether the pointer to the OD2TBL table is there. If the pointer is missing, it is put in.

Data A-text Q-routine identification elements are generated for each Q-routine and placed on the Data A-text data set. These indicate that the Q-routines are to be executed during initialization processing at execution time.

#### PROCESSING ERRORS

As phase 22 processes Data IC-text and ATF-text, a check is made of the clauses to be sure that they are allowed to be used together.

EBCDIC names for keys for table handling (prefix 01 or 02) are entered into the SRCHKY table by routine READF4 while the dictionary is being built. This table is used for syntax checking whenever the names are encountered.

#### BUILDING TABLES FOR LATER PHASES

Phase 22 builds eight tables for later phases. In addition, it uses the VALTRU table for syntax checking of the VALUE IS SERIES clause, but then leaves that table in storage for phase 3. The VALTRU table is built by phase 20 and described under that heading.

The QVAR and QFILE tables are built during Q-routine generation and stored for use by phase 3; the QFILE table may be updated by phase 21. They are discussed above under "Q-routine Generation."

During dictionary preprocessing (described above), routine SRH200 creates the INDKEY table.

The FDTAB table is built for phase 21.

If the SYMDMP or TEST option is in effect, phase 22 primes and builds as many as four tables, depending on the clauses in the source program, for phase 25: the OCTBL, HASTODO, VARLTBL, and RENAMTB tables.

PHASE 21

Phase 21 (IKFCBL21) is the last of the five phases that process the Data Division. When phase 21 is loaded into storage, the dictionary is complete except that the FD and SD entries and the LINAGE-COUNTER special register are dummy entries without data attributes written by phase 22.

The input to phase 21 includes Data IC-text, Data A-text (with two-byte prefix), E-text, and four tables. The PIOTBL table, built by phase 1B, and the FDTAB table, built by phase 22, are used to supply information about files for Data A-text. The QFILE table, built by phase 22, is updated if DCBs are created for checkpoint files. The CKPTBL table, built by phase 10, supplies information about each RERUN statement in the source program. Phase 21 uses the CKPTBL table to build the RUNTBL table, which is used in phase 51 to process verbs for RERUN files.

After phase initialization, each record is read from SYSUT3 and the action to be taken is determined. For a Data A-text element, the 2-byte prefix attached in phase 22 is removed, and the element is copied onto SYSUT4; FD and SD elements are selected for processing by phase 21; all other records are copied unchanged onto SYSUT4.

From the FD and SD Data IC-text elements and from information stored in the dictionary and the PIOTBL and FDTAB tables, phase 21:

- Completes FD dictionary entries from source program file description entries in the File Section.
- Completes SD dictionary entries from source program sort description entries in the File Section.
- Completes LINAGE-COUNTER entries.
- Writes the Data A-text for DCBs, DECBS, and buffers onto SYSUT4.

Phase 21 also produces E-text. If an error (E) level message is generated and the CSYNTAX option is in effect, the SYNTAX option is forced into effect and the options suppressed when SYNTAX is in effect are turned off (see "Compiler Options" in the section "Introduction").

FD DICTIONARY ENTRIES

The FD dictionary attributes work area (CI) is filled in from Data IC-text in FD form and from FDTAB table information in FD form. Routine FSTXT and its subroutine ACCMET do most of the processing. The remaining information needed, that is, base locator number, Q-Routine indication, maximum record size, and recording mode fields, is filled in by the FST000 routine.

The count and major code fields in the dictionary entry are predetermined by the Data IC-text. The access method field is filled in from the information in the access and organization fields of the Data IC-text.

As phase 21 sets up DCB, DECB, and FIB identifying elements for the file (described in "Data A-text Elements" in this chapter), subroutine ACCMET assigns DCB and DECB identifying numbers from the fields DCBCTR or DECBCT in COMMON and subroutine AMTXT assigns FIB identifying numbers from the AMICTR field in COMMON. These numbers are placed in the dictionary attributes work area.

If the checkpoint bit in the Data IC-text is set to 1, phase 21 searches the CKPTBL table for the entry for the file-name. The TYPE field in the CKPTBL table is tested and if the "END OF REEL/UNIT" option was specified, phase 21 builds a BSAM DCB for the checkpoint file and makes one entry in the exit list. See "Creation of Exit Lists" in this chapter. If the "integer-1 RECORDS" option was specified, phase 21 builds a BSAM DCB for the checkpoint file for each unique external name and creates a RUNTBL table entry from the CKPTBL table. In both cases, the Chain Pointer field in the CKPTBL table is tested and if it is not zero, the other entry for the file-name is processed as described above. After the RUNTBL table is completed, the CKPTBL table is released and the RUNTBL is made static. Finally, the CKPCTR cell in COMMON is set to the number of RERUN files with the "integer-1 RECORDS" option.

If DCBs are generated for checkpoint files and the QFILE table is present, phase 21 increments the DCB numbers in the QFILE table by the number of DCBs created for checkpoint files.

While forming partial FD entries, the ACCMET subroutine begins building the SAMETB and SMRCDTBL tables. Routine FST000 completes the fields in the tables.

Phase 21 builds the FD dictionary entries, the File Information Block (FIB), and the IND2TBL table entry if Record key is specified for VSAM files.

In building the dictionary entries, FIB, and IND2TBL table entry, phase 21 uses FS-text as its input.

#### SD DICTIONARY ENTRIES

The SD dictionary attributes work area (CI) is filled in from Data IC-text in SD form and from FDTAB table information in SD form. Routine SDTEXT fills in count and major code information, which is predetermined by the Data IC-text. Record lengths, BLL information, Q-Routine indication, and recording mode are filled in by the FST000 routine which does most of the processing for the entry.

#### LINAGE-COUNTER ENTRIES

LINAGE-COUNTER entries are completed from the Data IC-text and the dictionary.

#### DATA A-TEXT ELEMENTS

Phase 21 generates the following types of Data A-text elements from which phase 6 or 64 creates object text for the data area of the object module.

- Block address elements
- Constant definition elements
- Address constant definition elements
- DCB, DECB, and FIB address elements

To create these elements, phase 21 uses the COMMON area, the FDTAB table, Data IC-text, and the PIOTBL table from phase 1B. If there is an entry for a file in the PIOTBL table, the Data IC-text for the file has a pointer to the table. The PIOTBL table indicates the usage of the file. In addition, phase 21 creates the SAMETB, and SMRCDTBL tables to aid in producing Data A-text.

#### FIB ADDRESS ELEMENT

Phase 21 creates a FIB address element for each FD entry that describes a VSAM file.

#### BLOCK AND WORKING-STORAGE SECTION ADDRESS ELEMENTS

Phase 21 creates a block address element for each FD entry that describes a basic access method (BSAM, BISAM, and BDAM) or a queued access method (QSAM and QISAM) with the SAME AREA clause.

The element contains the location of the specified area in the object module data area (that is, the current value of field LOCCTR in COMMON) and the first base locator number assigned to the area.

#### CONSTANT AND ADDRESS CONSTANT DEFINITION ELEMENTS

Constant, address constant, and virtual reference definition elements are created to build exit lists and to specify the contents of DCBs and DECBs created for files described in FD entries.

#### Creation of Exit Lists

Exit lists are built of constant and address constant definition elements.

The area for the exit list is reserved and the pointer to the list is indicated in the DCBEXLST field. Some of the actual entries to the exit list will be made at object time by code generated by phase 51.

The format and contents of the exit list are described in "Appendix B. Object Module."

#### DCB'S AND DECB'S -- ADDRESS AND CONSTANT DEFINITION ELEMENTS

#### File Information Block (FIB)

Phase 21 creates the fixed portion of the File Information Block (FIB) for VSAM files. The FIB work area is generated by means of the GENFIB data macro. The AMTXT, IDTXT, and FST000 routines fill in fields



of the FIB, and the BEGIN routine writes the FIB as Data A-text (constant definition).

Phase 21 produces Data A-text address elements and constant definition elements which are used to create DCBs and DECBs. The DCB and DECB address element specifies the location of the DCB or DECB; the constant definition, and address constant definition elements specify the contents of fields within the DCB or DECB. These Data A-text elements are used by Phase 6 or 64 to produce the actual DCBs and DECBs.

In addition to the location of the DCB or DECB, which is the current value (adjusted to fall on a doubleword boundary) of the LOCCTR cell in COMMON, the DCB and DECB address element contains the DCB or DECB number assigned to the file from cell DCBCTR or DECBCT in COMMON.

The contents of the DCBs and DECBs are produced in several stages as information becomes available. The following discussion shows how phase 21 provides the contents of some of the DCB fields. DCB and DECB formats are given in OS/VS1 System Data Areas and OS/VS2 System Data Areas.

When phase 21 encounters the Data IC-text for an FD entry, the FSTXT routines

specify the size of the DCB for each file and DECB for files with basic access methods (BISAM, BSAM, and BDAM). The size is determined from the access method. Using the Data IC-text information, phase 21 issues constant definition elements which contain the information to be placed into the fields of the DCB and DECB; for example, file-name and organization.

If the file contains a SAME AREA or SAME RECORD AREA clause, additional processing described in "Block and Working-Storage Section Address Elements" must be performed.

#### Determination of Buffer Area Size

Constant definition elements for the DCB and DECB entries relating to the size of the buffer area to be allocated for a file are created after appropriate calculation. In most cases, the compiler allocates all the buffer area statically at compile time. But in the case of QSAM or QISAM files, and only for those unaffected by a SAME AREA clause, the system allocates the buffers automatically at OPEN time.

BUFFER AREA FOR A SINGLE FILE: The size of the buffer area to be allocated for any given file is determined in general as follows:

If

$c_2$  = maximum number of characters in a block (taken from BLOCK CONTAINS [ $c_1$  TO]  $c_2$  CHARACTERS)

$r_2$  = maximum number of records in a block (taken from BLOCK CONTAINS [ $r_1$  TO]  $r_2$  RECORDS, or default = 1)

KEY =  $\begin{cases} \text{size of ACTUAL KEY} - 4, & \text{if direct BSAM with key or direct BDAM} \\ 0 & \text{in any other case} \end{cases}$

$r_1$  = size of record length field

=  $\begin{cases} 4 & \text{if RECORDING MODE V} \\ 0 & \text{in any other case} \end{cases}$

$b_1$  = size of block length field

=  $\begin{cases} 4 & \text{if RECORDING MODE V} \\ 0 & \text{in any other case} \end{cases}$

RCD = maximum data record size for the file (calculated by the compiler from the maximum size of the relevant level-01 entries)

Then

BLK = block size

=  $\begin{cases} c_2 & \text{if CHARACTERS option specified in BLOCK CONTAINS clause} \\ b_1 + (\text{KEY} + r_1 + \text{RCD}) * r_2 & \text{in any other case} \end{cases}$

And also if

BCB = buffer control block size

=  $\begin{cases} 16 & \text{for BISAM or QISAM} \\ 12 & \text{for QSAM if RECORDING MODE S} \\ 8 & \text{for QSAM if RECORDING MODE F, V, or U} \\ 0 & \text{if neither BISAM, QISAM, nor QSAM} \end{cases}$

N = total number of areas for file

= number of alternate areas + 1 (taken from RESERVE...ALTERNATE AREAS for QSAM or QISAM)

SR =  $\begin{cases} \text{RCD} + 32 & \text{for QSAM if RECORDING MODE S} \\ \min[\text{RCD}, \text{track capacity}] + 8 & \text{for BSAM or BDAM if RECORDING MODE S} \\ 0 & \text{in any other case} \end{cases}$

Then

S = total buffer area for file

= BCB + (BLK\*N) + SR

BUFFER AREA INVOLVING MORE THAN ONE FILE: The total amount of buffer space to be allocated depends, in addition, on the presence of SAME AREA and SAME RECORD AREA clauses. Three cases are shown below. There are  $n$  files  $f$ , where  $f$  ( $i=k+1, \dots, n$ ) use VSAM, BSAM, or BDAM, and where  $f$  ( $i=1, \dots, k$ ) use some other access method. All  $S$ ,  $RCD$ ,  $rl$ ,  $bl$ , and  $KEY$  are determined according to the formulas and definitions shown in the preceding paragraph.

Note: For automatically buffered files, as described above, the buffer area is allocated only if the file is currently open.

---

Case 1.

---

If SAME AREA was specified for all files

$$\text{total size} = \max_{1 \leq i \leq n} [S]$$

---

Case 2.

---

If SAME RECORD AREA was specified for all files

$$\text{total size} = \sum_{i=1}^k [S] + \max_{1 \leq i \leq n} [RCD] + \max_{k < i \leq n} [bl + rl + KEY]$$

where  $\max_{1 \leq i \leq n} [RCD]$  represents the shared record area size

and  $\max_{k < i \leq n} [bl + rl + KEY]$  represents the size of the shared

block length, record length, and key fields (if any) for the BSAM and BDAM files.

---

Case 3.

---

If neither of the above clauses was specified for any file

$$\text{total size} = \sum_{i=1}^n [S]$$

SAME AREA clauses are processed by the SAME routine using the SAMETB table; SAME RECORD AREA clauses are processed by the SAMER routine using the SMRCDTBL table.

CLAUSE COMPATIBILITY

As phase 21 processes Data IC-text, a check is made of the clauses to be sure that they are allowed to be used together. This section gives some examples of the checking that is performed.

The clauses that can be used in an FD entry depend upon the access method of the file. For example, RECORD KEY cannot be used with a physical sequential file.

Phase 21 contains a list of valid clauses for each access method. When a Data IC-text element for an FD entry is encountered, the access method for the file is determined. Then, as the entry is processed, the clauses used are checked against the list to determine that no invalid clauses for that access method have been used.

PHASE 25

Phase 25 (IKFCBL25) is loaded only if the SYNDHP or TEST option is in effect. The major functions of phase 25 are:

- Building the OBODOTAB table and writing it on the debug data set (SYSUT5) if the program contains any OCCURS...DEPENDING ON clauses
- Building the DATATAB table and writing it on the debug data set (SYSUT5).

The operations of phase 25 are described in Diagram 3, located with the foldouts at the back of this publication. The functions of the OBODOTAB and DATATAB tables when SYNDHP is specified can be found in IBM OS/VS COBOL Subroutine Library Program Logic. The tables are also used by the program product IBM OS COBOL Interactive Debug when TEST is specified.

## PHASE 25 PROCESSING FOR THE DEBUG DATA SET

To build the OBODOTAB and DATATAB tables, phase 25 uses the following tables passed from phase 22:

- The DICOT table, which contains information about the COBOL dictionary.
- The HASH table which is used in locating dictionary entries.
- The QITBL table, which contains a COBOL dictionary pointer for every object of an OCCURS...DEPENDING ON clause.
- The ORTN table, which contains a COBOL dictionary pointer for every subject of an OCCURS...DEPENDING ON clause.
- The RENAMTB table, which associates renamed data-names with their renamers.
- The OCCTBL table, which contains information about each subject of an OCCURS clause.
- The MASTODO table, which identifies all data-names which do not contain an OCCURS...DEPENDING ON clause themselves, but one of whose subordinate items at the next level does.
- The VARLTBL table, which contains an entry for each variable-length item.

There is a DATATAB entry for each data item in the Data Division. There is also a DATATAB entry for some of the compiler-generated names associated with the Report Writer feature. There is an OBODOTAB entry for each unique object of an OCCURS...DEPENDING ON clause.

The OBODOTAB and DATATAB tables list the characteristics of data items in the Data Division. (See "Section 5. Data Areas" for the format of the OBODOTAB and DATATAB tables).

Entries for either the OBODOTAB or DATATAB table are built in a work area (WRKAREA) in phase 25. Each entry in the OBODOTAB and DATATAB tables is moved directly into the debug data set buffer as soon as it is completed. OBODOTAB entries are entered in the debug data set on fullword boundaries. DATATAB entries are not aligned.

Certain of the DATATAB entries contain pointers to OBODOTAB entries. Each DATATAB entry for the subject of an OCCURS...DEPENDING ON CLAUSE contains a pointer to the OBODOTAB entry for its corresponding object. The pointer consists of the relative block number within the OBODOTAB table and the displacement into the block (in fullwords). Each DATATAB entry for a data-name subordinate to the subject of an OCCURS...DEPENDING ON clause also contains a pointer to the OBODOTAB entry for the object of that OCCURS...DEPENDING ON clause.

Building the OBODOTAB Table

Phase 25 uses the OCCTBL table passed from phase 22 and builds the ODOTEL table in the process of building the OBODOTAB table. Phase 25 processing for the OBODOTAB table is shown in Figure 11.

After the OBODOTAB table is built, OCCTBL table entries are completed. The completed OCCTBL table is used by routine TESTSUBS during the building of the DATATAB table later in the phase.

Building the DATATAB Table

The BEGPASS routine controls building of the DATATAB table entries, using the SYMDICT DSECT. It performs the following functions:

- Calls LOCNXT to read dictionary entries
- Calls GETDEF to get generated card number for data-name from DEF-text. RENAMES items are ignored.
- Calls BLDRD to process RD level entries.
- Calls SETNAM to build fixed portion of entry. SETNAM calls PROCESLD to build variable portion of entry for LD under

FD, SD, Working-Storage, Linkage Section.

- Calls PROCENM to process RENAMES items for data-name. PROCENM calls ENTRDATA to move complete entry in output buffer.
- Branches to TESTSUBS to determine subscribed items. TESTSUBS uses the OCCTBL table for subscribing information, and calls ENTRDATA as above.

ENTRDATA routine calls WRITE5 to write buffer on SYSUT5 at end of buffer. PHASEND routine releases tables and repositions SYSUT4 when the dictionary processing is complete.

Figure 11. Building the OBODOTAB Table

**1** The COBOL source program contains both OCCURS and OCCURS... DEPENDING ON clauses.

```

77 X USAGE IS COMP PIC 9(3).
77 Y USAGE IS COMP PIC 9(3).
01 A.
02 B OCCURS 2 TIMES.
03 C OCCURS 10 TIMES DEPENDING ON X.
04 D OCCURS 10 TIMES DEPENDING ON Y.
05 E PIC 9.
    
```

**2** Phase 22 builds the OCCTBL table initializing it with the dictionary pointer of each subject of an OCCURS or an OCCURS... DEPENDING ON clause. A bit is set to indicate the latter.

OCCTBL Table

Displ	Dictionary Pointer for	Length for	Information for	ODO SW	To be filled in by Phase 25
100	B		B	OFF	
108	C		C	ON	
118	D		D	ON	

**OCCTEST**  
Tests for presence of OCCURS... DEPENDING ON clause

**3** If there are ODOs, routine ODOBLD builds the ODOTBL table by:

**4** Checking the ODOCT counter defined in phase 25 for the number of the ODO entry to be processed.

ODOCT counter

1

**ODOBLD**  
Builds the ODOTBL table

**5** Using that number to find the QRTN table entry for that OCCURS... DEPENDING ON clause.

QRTN Table

1st entry					0	0	0	6
2nd entry					0	0	0	8

QITBL Table

Dictionary pointer to attributes of ODO clause	Displacement of ODOTBL entry associated with ODO clause
	0 0 1 0
	0 0 0 2
Pointer for Y	0 0 0 8
	0 0 0 4
	0 0 1 2
Pointer for X	0 0 0 6
	0 0 1 4

**6** The QRTN table entry is used to find in the QITBL table the dictionary pointer for the object of the ODO clause.

ODOTBL Table

Dictionary pointer to attributes of object of ODO clause	Displ'mnt in OCCTBL to pointer of corresponding entry	For OBODOTAB pointer
Pointer for X	116	
Pointer for Y	126	

**7** The dictionary pointer for the object of ODO is inserted into the ODOTBL table.

**8** Routine BLDOBODO builds the OBODOTAB table on the Debug data set from information stored in the dictionary, using entries in the ODOTBL table to locate the data items which comprise the OBODOTAB table. BLDOBODO inserts a pointer (that is, the block number and the displacement within the block) to each OBODOTAB entry into its corresponding ODOTBL entry.

**BLDOBODO**  
Builds the OBODOTAB table; completes the ODOTBL table

OBODOTAB Table

0-012  
0-020

ODOTBL Table

Dictionary pointer to attributes of ODO clause	Displ'mnt in OCCTBL to pointer of corresponding entry	For OBODOTAB pointer
Pointer for X	116	0012
Pointer for Y	126	0020

**ENDP1**

**9** Finally, using the ODOTBL table, routine ENDP1 fills in the OBODOTAB pointer for each OCCTBL table entry which is the subject of an ODO clause.

OCCTBL Table

Displ	Dictionary Pointer for	Length for	Information for	ODO SW	Pointer to OBODOTAB for entry
100	B		B	OFF	for Displ = 116
108	C		C	ON	0-012
118	D		D	ON	0-020

Displ = 126

### PHASE 3

By the time phase 3 (IKFCBL30) is loaded into storage, almost all information on source program names in P0-text has been concentrated in the attributes of dictionary entries. Supplementary information is stored in the QVAR, QFILE, INDKEY, and VALTRU tables. Phase 3 can now replace each name with its attributes, as well as add information to verb strings such as SEARCH and OPEN.

Phase 3 also performs any other processing that requires the dictionary. In this manner, storage space for the dictionary and for dictionary ACCESS routines is freed for subsequent phases. Phase 3's use of the ACCESS routines and descriptions of their functions are given in "Appendix A. Table and Dictionary Handling."

Phase 3 processing thus consists of four main operations, all dependent on the dictionary:

- Building a Data Division glossary of all source program data-names.
- Replacing source program names with their attributes.
- Performing special processing on procedure-names in segmented programs, verb strings, and verb strings with CORRESPONDING options.
- Performing any syntax analysis that requires the dictionary.

Diagram 4 shows the overall flow of phase 3 operations. Phase 3 input consists of P0-text and E-text on SYSUT2, and the dictionary and the QFILE, QVAR, INDKEY, IND2TBL, and VALTRU tables in storage. Its output consists of P1-text and E-text on SYSUT3, DEF-text on SYSUT4, the glossary on SYSPRINT, and the DTAB table in storage.

After the PHINIT routine receives control from phase 00 and performs initialization for the phase, operations occur in two stages: glossary-building under the control of the GLOSRY routine, and translation of P0-text into P1-text under the control of the PHCTRL routine.

During the translation stage, special processing is performed on READ verb strings, OPEN verb strings; SORT and MERGE verb strings ADD, SUBTRACT, and MOVE verb strings with CORRESPONDING options; SEARCH verb strings; source program names and special registers; and syntax errors. If USE FOR DEBUGGING declaratives exist in the source program, additional special processing is performed on procedure names and ALTER verb strings.

### GLOSSARY BUILDING

The PHINIT routine determines from the SYM bit of the PHZSW1 switch in COMMON whether a glossary has been requested. If it has not, the PHINIT routine tests the APPWRO switch in the SWITCH1 cell in COMMON to determine if APPLY WRITE-ONLY was specified in the COBOL source program. If it was, the PHINIT routine branches to the TSTWRO routine, which scans the File Section entries of the dictionary. For each file definition entry in the dictionary in which the WRITE ONLY switch is on, the TSTWRO routine sets the major code to 7 in all the data-name entries associated with the file. When all the files have been processed, the TSTWRO routine branches to the GLORET routine, which initializes the translation stage of processing, reads in the first block of P0-text, and branches to the PHCTRL routine. If APPLY WRITE-ONLY was not specified, the PHINIT routine branches to the GLORET routine, and processing proceeds as described above.

If a glossary has been requested, the PHINIT routine branches to the GLOSRY routine, which prints out the glossary on SYSPRINT and simultaneously performs the same function as the TSTWRO routine.

The GLOSRY routine scans the dictionary with the use of the DICND1 field in COMMON (pointing to the last Procedure Division entry created by phase 1B, or UPSI entry created by phase 22) and the DICND2 field (pointing to the last Data Division entry created by phase 22). As each data-name is encountered in the dictionary, it is placed in location PRLINE along with pertinent information from its attributes. Phase 00 is then called to print PRLINE. When necessary, the GLOSRY routine converts numbers in the attributes from one mode to another.



TRANSLATION FROM P0-TEXT TO P1-TEXT

Before the GLORET routine branches to the PHCTRL routine for the translation stage of phase 3 operations, it stores the address of the first P0-text element in location PNTIN. The GETNXT routine moves the identification code of the element (the first halfword) into location GOTTEN. The PHCTRL routine then tests GOTTEN to determine the processing that should be performed.

If the element is a READ or RETURN verb, the PHCTRL routine calls the READFN routine to insert the appropriate record-name after the file-name. If the element is an ADD, SUBTRACT, or MOVE verb followed by an element for CORRESPONDING, the PHCTRL routine copies out the entire statement as P1-text, using the SEARCH routine to determine the uniqueness of the operand. It uses the CORRTRN routine to break down the statement into simple statements, each containing one of the matching pairs of elementary items.

If the element is a source program name, the PHCTRL routine calls the SEARCH routine to determine whether it is unique and then calls the GENOP routine to replace the name with its dictionary attributes and write it out as P1-text. If the name is a special register, however, the SEARCH routine generates the appropriate P1-text element.

If the PHCTRL routine encounters a SEARCH verb, it calls the STSRCH routine. If the element is a source card number, the PHCTRL routine places the number in CARDNO and then writes the element out unchanged on SYSUT3. All remaining elements are also written out on SYSUT3 unchanged.

If the element is a file-name in an OPEN verb string, the PHCTRL routine sets a switch for the GENOP routine to add information for label and error processing to the string.

If the element is a VSAM file-name, the FILENM routine initializes the Key Clause workarea. This information is used by the PHCTRL routine to determine that the file specified in a subsequent KEY clause is a VSAM file and by the DATANM routine to determine that the data-name specified in the KEY clause was specified on a RECORD KEY data-name.

The CORRTRN and STSRCH routines each process the entire string associated with its special condition. They use the SEARCH routine to determine whether the names in the verb string being processed are unique.

All these routines use the GENOP routine to replace the names with their dictionary attributes and write them out as P1-text elements. The GENOP routine also generates DEF-text for procedure-names if necessary.

The processing routines perform any diagnostic analysis that requires the dictionary. When a routine detects an error requiring action parameters that only a subsequent phase can determine, it substitutes an error symbol for the element in error. When it detects an error condition for which it can provide an entire message, it calls the ERROR routine with appropriate error parameters before returning to the PHCTRL routine.

When the PHCTRL routine detects an end-of-file condition, it branches to the EOF routine, which releases tables and returns to phase 00.

READ Verb Strings

The READFN routine checks the next P0-text element in the input buffer after a READ or RETURN verb to determine whether it is a file-name. If it is not, the READFN routine writes the verb element unchanged on SYSUT3 and returns to PHCTRL. Phase 4 can detect the error without the dictionary.

If a file-name does follow the READ verb element, the next dictionary entry after the file-name entry is checked to determine whether it is a record-name. If it is, the GENOP routine is used to build a P1-text data-name reference element for the record and to write it out after the file-name. In the case of multiple records, the attributes of the longest record in the file are used. An error symbol is substituted for the record-name attributes if the dictionary entry following the file-name is not a record-name.

Statements with CORRESPONDING Options

The CORRTRN routine checks to determine whether the operands in the source statement are valid. It then matches the subordinate, lower level data-names defined within the source statement hierarchies, and writes a P1-text statement for each matching pair. This, in effect, breaks down the CORRESPONDING option source statements into a series of similar statements, which together accomplish the operations implied by the source statement.

Two sample CORRESPONDING P0-text statements are given in Figures 12 and 13 along with the resulting P1-text. The step numbers given to the left of these figures refer to the procedure sequence below.

Step 1: The PHCTRL routine writes out as P1-text the entire statement up to the next critical program break. The source statements are put out so that phase 4 can perform a syntax check on them.

<u>P0-text:</u>	ADD CORRESPONDING R TO K
<u>P1-text:</u>	
Step 1:	ADD CORRESPONDING R TO K
Step 5:	ADD R TO K
Step 5:	ADD R1 TO K1
Step 5:	ADD R2 TO K2
Step 5:	.
Step 5:	.
Step 5:	.
Step 5:	ADD Rn TO Kn
Step 6:	CORRESPONDING

Figure 12. P1-text Resulting from an ADD CORRESPONDING Option

Step 2: Operand-1 and operand-2 are checked to ensure that they are both group items and valid data-names. Various procedures are followed, depending on what the checks reveal.

If operand-1 is not an EBCDIC name or data operand, no more processing is done on the statement, and the next P0-text element is read in. Phase 4 can detect this type of error.

If operand-1 is not an EBCDIC name but is a data operand (for example, a literal), the CORRTRN substitutes an error symbol for the operand in P1-text, calls the ERROR routine to put out error text, and reads in the next P0-text element. Phase 4 can detect this error, but the error symbol indicates that phase 3 has already produced an error message.

If operand-1 is not a group item, error text is generated, an error symbol is substituted for the operand, and the next P0-text element is read in. Phase 4 cannot detect this error.

If operand-1 is valid, but operand-2 is neither an EBCDIC name nor a data operand, an error symbol is substituted for operand-2 and a P1-text string is produced as follows:

Verb  
Attributes  
Preposition  
Error symbol

<u>P0-text:</u>	MOVE CORRESPONDING A (1) TO B
<u>P1-text:</u>	
Step 1:	MOVE CORRESPONDING A (1) TO B
Step 5:	MOVE A (1) <sup>+</sup> TO B <sup>+</sup>
Step 5:	MOVE A (1) <sup>(</sup> TO B <sup>(</sup>
Step 5:	.
Step 5:	.
Step 5:	.
Step 5:	MOVE A (1) <sup>°</sup> TO B <sup>°</sup>
Step 6:	CORRESPONDING

Figure 13. P1-text Resulting from a MOVE CORRESPONDING Option

The word CORRESPONDING is written after the string, and then the next P0-text element is read in. The string tells phase 4 that no matching pairs were produced because of an invalid operand-2.

If operand-1 is valid but operand-2 is not a group item, then error text, a P1-text string containing an error symbol, and the word CORRESPONDING are all written out, and the next P0-text element is read in.

Step 3: Assuming both operands are valid, the subject (operand-1) hierarchy in the dictionary is scanned for corresponding items at the same relative level in the object (operand-2) hierarchy.

Since the source statement operands have already been checked by the dictionary handling routines, the CORRTRN routine knows which operand has the highest level dictionary pointer. Before initiating the object hierarchy search, it ensures that the subject hierarchy pointer is at a lower level than that of the object hierarchy. If this is not the case, the operand-2 group becomes the subject hierarchy. This

is done to optimize the scan, since the dictionary handling routines look for the latest entry first through the HASH table (see "Appendix A. Table and Dictionary Handling").

If a group item in the subject hierarchy does not have a matching name at the same level in the object hierarchy, the rest of the items in the group are skipped. This is done because there is no possibility of finding a match for any of the items in the group.

Step 4: The subordinate items in the hierarchies are checked for conformity to the source language regulations. (For example, does the item contain a REDEFINES or OCCURS...DEPENDING ON clause? If it does, ignore the item.) No error symbols or messages are generated unless no match is found for any of the subject hierarchy items. In this case, a P1-text string (verb, error symbol, preposition, error symbol) is written out to tell phase 4 that there were no matching items.

Step 5: When a correspondence is found, assuming both items are valid, P1-text statements similar to the source statement are generated.

Step 6: If there are no more corresponding items, a P1-text element for CORRESPONDING is written out, and the next P0-text element is gotten. The word CORRESPONDING tells phase 4 that the previous element was the last of a complete CORRESPONDING statement.

#### SEARCH Verb Strings

For each table to be searched there is an entry in the INDKEY table containing such information as the length of the table, as

well as pointers to attributes in the dictionary for all the data items associated with the table. The STSRCH routine adds some of this information and attributes to the SEARCH verb string.

The STSRCH routine first writes the verb element out on SYSUT3 and then examines the element that follows. This element should be an EBCDIC name. If it is not, the STSRCH routine abandons processing the text as a SEARCH string and returns to the PHCTRL routine to process it in the normal manner. No error text is produced, as phase 4 can detect the error.

If the element is an EBCDIC name, the STSRCH routine uses the SEARCH routine to determine that the name is unique. During its processing the SEARCH routine places the pointer to the dictionary entry for the name in location ID1PTR. The STSRCH routine uses this pointer as an argument to find an entry in the INDKEY table that contains the same pointer. This entry, in turn, contains pointers to entries in the dictionary for all the index-names, keys, and OCCURS...DEPENDING ON objects associated with this SEARCH verb string.

Figure 14 shows the P0-text input for a SEARCH format-1 verb string including a VARYING clause, along with the resulting P1-text output. Note that either a data-name or a literal may be inserted after the element for identifier-1 to express the length of the table. Also, depending on the type of EBCDIC name in the VARYING clause, any of three combinations of P1-text elements may be inserted into the string.

Figure 15 shows input and output for the SEARCH format-2 (SEARCH ALL) string.

Note that although in the source program the SEARCH statement may continue beyond the AT END through a number of conditional and imperative statements, the STSRCH routine stops processing before the AT END and returns to the PHCTRL routine to handle the remainder of the statement.

<u>Phase 1B Output</u>	<u>Phase 3 Output</u>	<u>Meaning of Element</u>	<u>Processing by Phase 3</u>
44   5E	44   5E	Verb, SEARCH format-1.	Copied out unchanged.
23   EBCDIC Name	30   Attributes	Data-name, identifier-1 (table to be searched).	Name replaced by its dictionary attributes.
	30   Attributes	Data-name, object of OCCURS...DEPENDING ON.	Attributes taken from dictionary, using pointer in INDKEY table entry that contains pointer to identifier-1 dictionary entry.
	OR		
	32   Literal	Represents maximum number of occurrences.	Literal taken from INDKEY table entry that contains pointer to identifier-1 dictionary entry.
54   88	54   88	VARYING	Copied out unchanged.
23   EBCDIC Name	36   Attributes	Data-name for index-name-1 that belongs to table.	Name replaced by attributes found in dictionary using pointer in INDKEY entry for identifier-1.
	OR		
23   EBCDIC Name		Data-name for index-name-1 that does not belong to table, if specified.	
	36   Attributes	Data-name for index-name-1 that belongs to table.	Name replaced by attributes found in dictionary using pointer in INDKEY entry for identifier-1.
	54   88	VARYING	Added for phase 4 convenience.
	36   Attributes	Data-name for index-name-1 that does not belong to table, if specified.	Name replaced by attributes found in dictionary.

Figure 14. P1-text Written for SEARCH Format-1 P0-text (Part 1 of 2)

<u>Phase 1B Output</u>	<u>Phase 3 Output</u>	<u>Meaning of Element</u>	<u>Processing by Phase 3</u>
OR			
[23 EBCDIC Name		Data-name for identifier-2 that does not belong to table, if specified.	
	[36 Attributes	Data-name for index-name-1 that belongs to table.	Name replaced by attributes found in dictionary using pointer in INDKEY entry for identifier-1.
	[54 88	VARYING	Added for phase 4 convenience.
	[30 Attributes	Data-name for identifier-2 that does not belong to table, if specified.	Name replaced by attributes found in dictionary.
[54 70	[54 70	AT	Copied out unchanged by PHCTRL.
[54 A1	[54 A1	END	Copied out unchanged by PHCTRL.
.	.		
.	.		
.	.		

Figure 14. P1-text Written for SEARCH Format-1 P0-text (Part 2 of 2)

Phase 1B Output	Phase 3 Output	Meaning of Element	Processing by Phase 3
[44 5F ]	[44 5F ]	Verb, SEARCH format-2.	Copied out unchanged.
[23 EBCDIC Name ]	[30 Attributes ]	Data-name, identifier-1 (table to be searched).	Name replaced by its dictionary attributes.
	[BB Literal ]	Literal representing number of keys.	Literal taken from INDKEY entry that contains pointer to identifier-1 dictionary entry.
	[30 Attributes ]	Attributes of first key.	Name replaced with dictionary attributes pointed to in entry containing pointer to identifier-1 dictionary entry.
	:	:	
	[30 Attributes ]	Attributes of last key.	Name replaced with dictionary attributes pointed to in entry containing pointer to identifier-1 dictionary attribute.
	[36 Attributes ]	First index-name attached to table.	Name replaced with dictionary attributes pointed to in entry containing pointer to identifier-1 dictionary attribute.
	[30 Attributes ]	Data name, object of OCCURS...DEPENDING ON.	Attributes taken from dictionary using pointer in INDKEY table entry that contains pointer to identifier-1 dictionary entry.
	OR		
	[32 Attributes ]	Literal representing maximum number of occurrences.	Literal taken from INDKEY table entry that contains pointer to identifier-1 dictionary entry.
[54 70 ]	[54 70 ]	AT	Copied out unchanged by PHCTRL.
[54 A1 ]	[54 A1 ]	END	Copied out unchanged by PHCTRL.

Figure 15. P1-text Written for SEARCH Format-2 P0-text

Determining the Uniqueness of a Name

The SEARCH routine moves the source program name it is to analyze from the input buffer to the location WKAREA. It then determines from the dictionary whether the name is unique. If it is, the SEARCH routine returns to the PHCTRL routine to replace the name with its dictionary attributes and put the result out as P1-text.

Special Registers: If the name is not in the dictionary, the SEARCH routine searches for it in the SPCREG area, which contains the names of special registers. Associated with each name is a pointer to dummy attributes in the REGATT area. The SEARCH routine replaces the special register name with its dummy attributes and calls the GENDAT routine to have them written as P1-text.

Qualifying Names: In P0-text, a name and its qualifiers are in reverse order of their appearance in the source program. When the SEARCH routine finds that a name is a qualifying name, it calls the QUALIF routine. The QUALIF routine searches the dictionary for each name in the string of qualifying and qualified names. If the qualified name is truly unique, the location of its attributes in the dictionary is returned to the PHCTRL routine. Its qualifiers are discarded.

Replacing Names with Dictionary Attributes

The PHCTRL routine calls the GENOP routine to replace a name with its dictionary attributes and then write the result out in P1-text format through the GENDAT routine.

Except for condition-names (which are never passed) and special registers (which contain zeros), the pointer to the dictionary entry itself is appended to these attributes. Although the dictionary does not exist after phase 3 operations, phases 50 and 51 use the pointer as an argument in syntax analysis, and phase 6 or 64 uses it as an identification code.

The GENOP routine determines from the attributes which kind of P1-text element should be generated. Special processing for particular types of names is described below.

Data-names: Data-name reference elements are generated for data-names. A unique data-name reference element is generated if dataname was specified in a USE ERROR DECLARATIVE with giving option. If the Q-Routine bit in the attributes is on, the

QVAR table pointer is used to locate the GN number to be added to the attributes. In the case of an elementary item, the GN number in the entry pointed to is used. In the case of a group item, the GN number in the entry for the next subordinate item is used.

File-names: File-name reference items are generated for file-names. If the Q-Routine bit in the attributes is on, the QFILE table is searched, using the pointer in the attributes, for a GN number to be added to the attributes.

If the OPENS location contains a 1 (set by the PHCTRL routine when it encounters an OPEN verb element), the GENOP routine searches for the GN numbers following the file-name element for label and error processing. These numbers are inserted between the attributes and the dictionary pointer in the resulting P1-text element.

If the file is a VSAM file, a VSAM file-name reference element is generated.

CD-names: CD-name reference items are generated for file-names. If the Q-Routine bit in the attributes is on, the QFILE table is searched, using the pointer in the attributes, for a GN number to be added to the attributes.

Procedure-names: When the name is a procedure-name reference in a segmented program, GENOP routine searches the dictionary for the section in which the name is defined. It then adds the priority number of the section to the attributes of the procedure-name reference.

Condition-names: When the GENOP routine encounters a condition-name, it creates a P1-text string that associates the item with the values for which it is to be tested. It uses pointers in the dictionary attributes of the condition-name to find the dictionary attributes of the item, as well as the test values in the VALTRU table. Figure 16 shows the P0-text input and P1-text output for a condition-string without a VALUE...THRU clause. Figure 19 shows the P0-text and P1-text for a condition-string with a VALUE...THRU clause.

Debugging

DEBUG-ITEM references are only valid in USE FOR DEBUGGING declaratives. During phase 3 processing when the last such declarative has been processed (END DECLARATIVES or non-USE FOR DEBUGGING declaratives encountered) the dictionary entries for a

DEBUG-ITEM are invalidated (ENDDBG routine). Further references will result in diagnostics. The ENDDBG routine invalidates dictionary references to fields of the DEBUG-ITEM special register by inserting an invalid character as the first character of each debugging special register in the dictionary. ENDDBG will guarantee that Procedure Division references to any DEBUG-ITEM field will be flagged.

If ALL PROCEDURES was specified in any USE FOR DEBUGGING declarative, DTAB is primed and DTAB entries created for all procedure-names encountered, except those in USE FOR DEBUGGING declaratives. PN definition handling is processed by the procedure-name handling routine (PRONAM). If the program has WITH DEBUGGING MODE specified, and one of the debug declaratives has specified ALL PROCEDURES, and this PN definition is not a debug procedure, then build and add an entry to the DTAB for phase 35.

The ALTSCAN routine appends a BCD literal element after each target PN in an ALTER statement. Phase 35 uses this routine to build a debug verb for ALTER procedure-names. ALTSCAN will loop until an error or all clauses of the ALTER statement are processed. Process for each iteration:

- Expects first P0 element to be PN1
- Checks for and skips by TO PROCEED TO

- Expects next P0 element to be PN2.
- If valid ALTB CD, builds and generates BCD literal. For PN2 after PN2 in P1-text.
- Any deviation causes exit from ALTSCAN

#### Error Processing

The processing routines branch to the ERROR routine when E-text for a complete diagnostic message can be generated. The parameter list following each branch consists of the message number, the severity code, a count of the parameters if any, and the addresses of the parameters. The ERROR routine builds E-text for a message in location ERMSG, calls phase 00 to write it out on SYSUT3 along with the P1-text, and then returns to the calling routine. Phase 3 also sets the ERRSEV cell in COMMON to the highest error severity level encountered. If an error (E) or disaster (D) level message is generated and the CSYNTAX option is in effect, the SYNTAX option is forced into effect and the options suppressed when SYNTAX is in effect are turned off (see "Compiler Options" in the chapter "Introduction"). The format of E-text is shown in "Section 5. Data Areas" and the manner in which diagnostic messages are later generated from it is described in the chapter "Phases 70, 71, and 72."



<u>Phase 1B Output</u>	<u>Phase 3 Output</u>	<u>Meaning of Element</u>	<u>Processing by Phase 3</u>
[54 07	[54 07	IF	Copied unchanged by PHCTRL.
[23 EBCDIC Name		Condition-name	Uses pointer in dictionary entry for condition-name to find entry for elementary item.
	[30 Attributes	Elementary item	Writes elementary item attributes from its dictionary entry.
	[50 06	EQUALS	GENOP generates.
	*  Literal	First value to be tested.	Taken from VALTRU table entry pointed to in condition-name attributes.
	[54 54	OR	GENOP generates.
	[50 06	EQUALS	GENOP generates.
	*  Literal	Last value to be tested.	Taken from VALTRU entry pointed to in condition-name attributes.
Imperative statement	Imperative statement		GENOP returns to PHCTRL to handle remaining processing.
*Code identifying type of literal.			

Figure 16. P1-text Written for Condition-String Testing Multiple Values without Using the VALUE...THRU Clause

Phase 1B Output	Phase 3 Output	Meaning of Element	Processing by Phase 3
54 07	54 07	IF	Copied unchanged by PHCTRL.
23 EBCDIC Name		Condition-name	Uses pointer in dictionary entry for condition-name to find entry for elementary item.
	30 Attributes	Elementary item	Writes elementary item attributes from its dictionary entry.
	54 5C	NOT	GENOP generates.
	50 0A	LESS THAN	GENOP generates.
	* Literal	First value in THRU option.	Taken from VALTRU entry pointed to in condition-name attributes.
	54 5D	AND	GENOP generates.
	54 5C	NOT	GENOP generates.
	50 08	GREATER THAN	GENOP generates.
	* Literal	Second value in THRU option.	Taken from VALTRU table entry pointed to in condition-name attributes.
Imperative statement	Imperative statement		GENOP returns to PHCTRL to handle remaining processing.
*Code identifying type of literal.			

Figure 17. P1-text Written for Condition-String with VALUE...THRU Clause

Phase 35 (IKFCBL35) processes the Procedure Division for debugging. It first scans the Declaratives Section for USE FOR DEBUGGING verbs and their operands, then adds to the source program's text, the verb necessary to cause invocation of the USE FOR DEBUGGING declaratives. Phase 35 is invoked only if WITH DEBUGGING MODE is specified, and at least one USE FOR DEBUGGING declarative is present at completion of processing, all tables are released and control is returned to phase 00.

Phase 35 is invoked by phase 00 to process USE FOR DEBUGGING declaratives, verifying each operand. Phase 35 scans Procedure Division IC-text for data items for which debugging is specified, and generates debugging text for processing by later phases.

IKFCBL35 first scans the Declaratives Section for USE FOR DEBUGGING verbs, analyses each verb, and makes an entry in the DTAB for each valid operand. If at least one valid operand exists in a USE FOR DEBUGGING sentence, a USE FOR DEBUGGING verb is generated into the P1A-text. Upon reaching the end of the debug declaratives, the DTAB is complete and the rest of the program is scanned for references to USE FOR DEBUGGING operands.

Two types of debugging verbs are generated as references to an operand of a USE FOR DEBUGGING declarative are encountered:

1. The DEBUG transfer of control verb consists of a verb code and an option byte. The option byte indicates the type of transfer of control. Phase 35 generates DEBUG transfer verbs for the situations listed below. If at least one procedure-name is a USE FOR DEBUGGING operand, then:

- Prior to a procedure-name definition, if it is a USE FOR DEBUGGING operand, but not if it is a declarative section, or if it is preceded by a COBOL conditional sentence, unconditional GO statement, END-DECLARATIVES control break, or an EXIT verb.
- Prior to a GO statement
- Prior to an I/O and/or conditional sentence.

- Following a conditional sentence.

2. The DEBUG verb that later causes a call to the DEBUG subroutine (ILBOBUG) to be made. Each call to ILBOBUG causes at least one DEBUG declarative to be invoked. The operands of this verb are the procedure-name number and priority number for the declarative to be invoked, an alphanumeric literal which will be the contents of the debug-name, a dictionary attributes item (dummy, if a DEBUG verb is produced for a procedure-name), or an alphanumeric literal giving the contents of DEBUG-contents, and an optional byte giving additional information about the USE FOR DEBUGGING operand reference.

When specified as a USE FOR DEBUGGING operand, the following items cause a DEBUG verb to be generated:

- Any explicit reference to a QSAM or VSAM file-name, or a CD-name.
- A Procedure-name reference that is the first operand of an ALTER verb.
- A procedure-name definition.
- Any explicit reference to an identifier when specifying, ALL REFERENCES...; otherwise, only when the identifier is changed.

**Note:** Certain verbs, because of where the language specifies the declarative to be invoked, requires special processing.

In general, the DEBUG declaratives are invoked prior to the execution of a procedure-name and after the execution of a verb.

Special processing is required for any USE FOR DEBUGGING operand identifier that is indexed or subscripted, so that it may be properly handled by phase 50. A special bit is set on in the attributes indicating that this identifier has both subscripting and debugging. Also, a DBGSS (debugging subscript) verb string is built and generated preceding P1-text containing any such USE FOR DEBUGGING operands.

Source output (P1-text) and DEBUG output are accumulated in separate tables until the proper time (usually end-of-verb) when both tables will be generated to P1A-text.

**Note:** At any given generation moment, only one DEBUG verb will be put to P1A-text per unique USE FOR DEBUGGING operand referenced, unless the operand is a subscripted or indexed identifier.

Some strings of debugging text are produced twice (for example, when a declarative must be invoked on the true and false paths of a condition).

#### TABLE HANDLING

**P1TEXT:** Phase 35 creates this table and uses it to accumulate each verb string until debug processing for it has been completed, at which time the text in this table is generated. Phase 35 deletes P1TEXT upon completion of processing.

**DTAB:** Phase 35 builds or adds to the DTAB table, saving the data descriptions of all valid operands found in the USE FOR DEBUGGING sentences. Upon completion of building DTAB (at end of DEBUG declaratives or END declaratives) it scans the remaining P1-text for operands that match the entries in DTAB. Whenever a match occurs, information from the DTAB entry is used to build a debug verb for the operand. Phase 35 deletes DTAB upon completion of processing.

**DBGTXT:** Phase 35 creates this table and uses it to accumulate DEBUG verb text while processing an input verb string. Phase 35 deletes DBGTXT upon completion of processing.

**VRBDN:** Phase 35 creates this table and uses it to describe each data item encountered by the current input in P1-text verb string. Phase 35 deletes VRBDN upon completion of processing.

**PH35VRBS:** An internal table used when analyzing a verb string for USE FOR DEBUGGING operands. PH35VRBS defines the proper syntax analysis routine for a specific verb, and its initial process control flag settings. Each entry in PH35VRBS contains the COBOL code for the verb, initial analysis process control flag settings, address of the verb analysis subroutine (VRBANALZ), and the length of the entry.

When phase 35 encounters a verb in the P1-text input stream, the PH35VRBS table is searched for the proper entry corresponding to the verb. The entry, with its initial settings is then copied into a work area, VRBINFO, which is utilized by phase 35 during verb analysis.

#### PROCESSING ROUTINES

**PHCTRL:** The Phase Controller routine (PHCTRL) controls the processing of phase 35. Upon entry to phase 35, PHCTRL performs the following initialization functions:

- Prime the DTAB table if not previously done by phase 30
- Prime the P1TEXT table
- Prime the DBGTXT table
- Prime the VRBDN table

**PH35INIT:** A subroutine of PHCTRL, initializes processing, including Tamer table priming. PHCTRL transfers control to the ANLZUFDS routine, which analyzes the USE FOR DEBUGGING sentences and builds a DTAB table entry for each valid operand. When control is returned to PHCTRL (no more USE FOR DEBUGGING sections), it will determine the category of the next input element and transfer control to one of two routines; GOTAVERB (processes all verbs encountered), or PNDEFRTN (checks a PN definition for debugging and may generate a debug transfer verb as well).

**ANLZUFDS:** A subroutine of PHCTRL, is called to scan the entire verb sentence for USE FOR DEBUGGING declaratives (P1-text) for proper operands until the end of all debug declaratives, or END DECLARATIVES is found. The VRBANALZ routine is used to handle syntax variations, GETDI is used to build the necessary VRBDN entries and debug text, and GENTXT is used to generate the contents of the P1TEXT and DBGTXT tables to output. The P1-text is read until the first section PN definition in the declaratives is found. For each USE FOR DEBUGGING declarative, the USE FOR DEBUGGING sentence is examined. A USE FOR DEBUGGING statement must begin with a section definition. Each operand is checked for validity (for example; not RD, index-name or special register). ANLZUFDS invokes CKUFDOPS for each USE FOR DEBUGGING declarative. In turn, CKUFDOPS invokes CKUFDOP for each operand in the sentence. CKUFDOP verifies the operand as valid and unique (QSAM or VSAM file-name, CD-name, PN, data-name), then adds an entry to the DTAB table via the DTABADD routine. If a PN, the BGALLPRC bit in COMMON must be tested to verify that ALL PROCEDURES had not been specified. If so, the PN must be diagnosed as invalid and discarded. A PN will be entered as FF, plus the PN number in the DICTPTR field of the DTAB entry. For a non-PN, the DICTPTR field contains the dictionary pointer. A special check must be made against each identifier to

ensure that PAGE-COUNTER, LINAGE-COUNTER, LINE-COUNTER, or PRINT-SWITCH have not been used as an operand of the USE, if so, they are diagnosed and discarded.

Once it has been determined that an identifier is valid, the size of the operand must be compared to the MAXBGITM cell in COMMON. At the end of phase 35, MAXBGITM will contain the size of the largest element for which debugging is validly requested. (Maximum allowed, however is 32K-57.)

If at least one operand in the sentence is valid, a USE FOR DEBUGGING verb is generated. Then input text, up to the next declarative section or END DECLARATIVES, is generated to output unchanged. When END-DECLARATIVES or a non-USE FOR DEBUGGING declarative section is encountered, control is returned to PHCTRL.

GOTAVERB: determines the verb and passes control to ANLZVRBS.

ANLZVRBS: ANLZVRBS is a subroutine of PHCTRL that processes complete P1-text verb strings for debugging. This subroutine searches each verb string looking for operands that were specified in a USE sentence. VRBANLZ routines are invoked to handle specific SYNTAX and processing variations. ANLZVRBS will find other verbs in the internal phase 35 table (PH35VRBS), and copy the initial VRBANALZ control flags from the table. (PH35VRBS defines the proper syntax analysis routine for a specific verb and its initial process control flag settings. Each operand produces only one debug verb for its use in a given verb string (except subscripted or indexed variables).

P1-text is kept in the P1TEXT table as it is being read. The debug verb text is also accumulated in the DBGTXT table as debugging operands are matched in the verb string. The P1TEXT table, and the DBGTXT table are generated when end of verb is determined (except for PERFORM, IF, ON, and SEARCH). End of verb is determined by searching for; card number, verb, or a special keyword (AT, END, NO (in RECEIVE), DATA, WHEN, THEN, or AFTER (in PERFORM)). When one of these occurs, the following is the sequence of the generation of tables:

- For verbs that end in an IF, UNTIL, or WHEN condition, or GO TO DEPENDING ON, DBGTXT is put out first, then P1TEXT.
- For special keywords (except THEN or AFTER (in PERFORM) or ATEND (in RETURN)), the P1TEXT table including special keywords, is generated, then DBGTXT.

- For special keywords THEN or AFTER (in PERFORM) DBGTXT table, then P1TEXT table excluding special keywords, is generated.
- For special keyword AT END (for RETURN) the P1TEXT table, excluding special keywords, is generated, then the DBGTXT table is generated.

VRBANALZ: The VRBANALZ subroutines invoked by ANLZVRBS at entry point based on the address in PH35VRBS. The major objective of each VRBANALZ subroutine is to process a given verbs P1-text string, locating and generating debug text for USE FOR DEBUGGING operands encountered, with minimal dependency on syntax analysis (VRBANALZ subroutines only perform as much syntax checking as necessary for location verification).

Some VRBANALZ subroutines look only for specific keywords that will give information about data items encountered before and/or after the keywords. Others may perform more complex operations.

Because of the generality of the VRBANALZ subroutines, it is possible to combine the processing of several similar verbs in one routine. Most of the VRBANALZ subroutines may be invoked several times while ANLZVRBS is processing a particular verb string. On each invocation, most will process what it recognizes, or copy what it knows is unimportant (such as extraneous COBOL words), then return to ANLZVRBS which may determine analysis for the current verb string if finished, and/or pass on the current P1 element to GETDI which checks if it is a USE FOR DEBUGGING operand.

Some routines process verbs whose debugging analysis is very sensitive to syntax. These routines often are invoked only once for a given verb occurrence and will stop analysis immediately if a syntax incongruity is encountered.

Input to VRBANALZ is the current P1 element addressed by P1TXTPTR (address of COMMON is in COSADR). Three sets of control flags (switches) are:

- PH35FLGS: phase control flags maintained continuously throughout phase 35.
- CURVFLGS: verb control flags. Original settings copied from PH35VRBS entry for the current verb and maintained continuously throughout the current verbs processing.
- ADDVFLGS: Additional verb control flags initialized prior to the current verb processing and maintained

continuously throughout the current verbs processing.

Output from VRBANALZ is P1-TEXT and DEBUG-TEXT saved in their respective Tamer tables, P1TEXT and DBGTEXT (some VRBANALZ subroutines) will generate these tables to P1A-TEXT when analysis deems it necessary. Input flags (switches) may be reset.

**PNDEFRTN:** The PNDEFRTN routine checks procedure-name definitions for debugging. PNDEFRTN determines if the PN definition is a USE declarative header. If so, any debug text for a PN definition is saved in the DBGTEXT table later to be generated following the P1-text for the USE verb. For any other PN definition, the debug text will be saved in the P1TEXT table. If this is any PN definition not immediately following an END DECLARATIVES, a conditional sentence, an EXIT verb, or an unconditional GO statement, a DEBUG transfer verb is generated. The PN definition element is saved in the P1TEXT table. Later, when PN analysis is complete, GENP1TT is called to generate the P1TEXT table, including the PN definition element, and possibly the DEBUG verb text for the PN definition.

**ERROR:** The ERROR subroutine builds and produces E-TEXT for error messages and associated parameters, and terminates compilation, if necessary. If error was disaster level, phase and compiler terminates, either immediately by ABEND, or shortly via EOFRTN.

**EOFRTN:** The EOFRTN routine receives control at end-of-file to release the tables and to terminate phase processing. EOFRTN guarantees that text tables have been generated, and releases all Tamer tables used by this phase.

#### NON-DEBUGGING DECLARATIVE CONSIDERATIONS

Collecting information about various data items encountered in a verb string as potential debugging operands is handled by the VRBDN table in phase 35. Each entry in the VRBDN table corresponds to a potential debugging operand in the verb string. Information stored includes whether the operand: 1) was found in the DTAB, 2) is unique, 3) is subscripted, or 4) will change value. Additionally, the entry identifies the offset in the DBGTX table where the debug verb, if any was built for this operand, is stored. When text is generated from the DBGTEXT table, phase 35 references each entry of the VRBDN table to determine if its corresponding debug verb should be generated and where in DBGTEXT it is located.

Four features of the debugging language make it necessary for phase 35 to be aware of the context in which the operand it is processing appears:

1. If an identifier is specified as an operand in a USE sentence without the ALL REFERENCES phrase, the declarative is to be invoked only if the identifier is changed. With the ALL REFERENCES phrase, the declarative is invoked whenever the identifier is explicitly referenced.

**Note:** When an identifier is encountered with certain verbs you cannot be sure if it will be changed until a later word is recognized in the verb (for example; RELEASE|REWRITE|WRITE. If FROM is specified, record-name has changed).

2. Phase 35 is sensitive to the syntax if the compiler has generated implicit text. Phase 35 must not generate debugging verbs for implicit text operands. The following verbs require special handling because of compiler-generated text:

- ADD|SUBTRACT|MOVE CORRESPONDING  
Only the first string is eligible for debugging. Subsequent ADD|SUBTRACT|MOVE verbs until CORRESPONDING (signifying end of string) are ignored.

- READ|RETURN  
Record name has been appended after file-name. Must be ignored.

- WRITE|REWRITE  
File-name has been inserted prior to record-name. Must be ignored.

- SEARCH  
After identifier-1 will be either identifier or literal. Ignore if identifier as it was generated.

- SEARCH ALL

a. When BBxxxx is encountered in input ignore the next xxx elements (keys for table).

b. Next two elements, index-name and identifier or literal must also be ignored.

- SORT|MERGE  
Phase 30 generates three SORT|MERGE strings as well as OPEN|CLOSE for USING|GIVING files. Only one invocation of the debug verb should be generated for each applicable operand only after the last string

generated for the SORT|MERGE statement.

3. Upon recognition of certain non-verb keywords to generate the appropriate P1TEXT table and DBGTYT table. The keywords are; AT, END, NO (in RECEIVE), DATA, WHEN, THEN, and AFTER (in PERFORM).

4. Subscripting or indexing. When an identifier is found which qualifies for debugging, the next operand must be checked to see if it is a left parenthesis. If so, a bit is set on in the attributes of an identifier to indicate that this subscripted identifier requires debugging. A verb will also be put out which will contain the actual subscript string. This will be produced prior to the source verb string.

Note: Debugging is only produced once for any operand in a given verb string, but if it is subscripted or indexed it will be invoked once for each occurrence of the subscripted variable in the verb string.

Output

There are three types of output produced by phase 35:

1. DEBUG transfer of control verb

4483	DEBUG transfer
2401nn	option byte

where nn has the following settings:

#GO	GO
#FALLTHRU	FALL THROUGH
#MERGE	MERGE

2. DEBUG verb:

448A	DEBUG verb
D0...	PN reference of
	USE FOR DEBUGGING
	declaratives
34...	DEBUG-NAME
26/21/25/30*/34...	DEBUG-CONTENTS
24nxxx**	option byte

\* identifier, if subscripted or indexed, a bit is on.

\*\* bit will be set on if this is the last DEBUG verb in a series.

xx has the following settings:

#IDCD	ID or CD
#ALTER	ALTER
#PN	PN
#FDREAD	FD for READ
#FDNREAD	FD not for READ
#LASTDBG	last DEBUG verb in a series

3. DEBUG subscript verb:

4496	
5200	left parenthesis
30....	identifier
.	
.	
5201	right parenthesis

Later phases use the DEBUG transfer verb to update the Task Global Table with DEBUG-LINE and type of transfer of control. These fields are used by the ILBOBUG subroutine when invoked for procedure-name to properly set up debug-items.

#### PHASE 4

Phase 4 (IKFCBL40) continues the transformation of a source program Procedure Division into machine-language instructions. Its main functions are:

- Transforms P1-text into P2-text
- Transforms P2-text into ATM-text for the UNSTRING verb
- Analyzes syntax and checking for errors in the P1-text statements.
- If COUNT is in effect, converts all verbs to Data A-text and defines block nodes with a counter in both the count table (Data A-text) and P2-text.

During phase 4, the card number of the statement currently being processed is kept in a three-byte cell labeled CARDNO, internal to phase 4, and in the CURCRD cell of COMMON.

#### TRANSLATION OF P1-TEXT TO P2-TEXT

Control flows through routine IDENT. This routine scans the current input element, anticipating either a procedure-name definition, a verb, or a program break. If the element is a procedure-name definition, control is given to routine IDLHN. If a verb is found, control is given to the verb analyzer routine for that particular verb. The verb analyzer processes the verb and its operands. Program breaks are processed by IDBRK, a subroutine of IDENT.

These routines return control to IDENT, with the input pointer containing the address of the next P1-text element.

#### VERBS

There is a separate verb analyzer routine for each COBOL verb.

The verb analyzer routines use a number of tables while building a verb string.

The STRING table is used by most verb analyzers that produce output. It holds an output string while it is being built. The string is held in the table, rather than being put out in parts as it is built, because:

- A string is not issued unless it is free of errors. This cannot always be determined until the entire string is produced.
- Sometimes the information that appears at the end of a P1-text statement (for example, UPON CONSOLE in a DISPLAY statement) is put at the beginning of the string as an aid to phase 51.

A string of P2-text is put out with a maximum of five operands. If more than five operands are required by a single verb, a continuation string is put out. See the discussion of the DISPLAY string. For an example of a continuation string, see the discussion of the DISPLAY statement in the chapter "Phase 51."

When phase 4 encounters an UNSTRING verb, it first puts out P2-text for the verb. All other information is put out on SYSUT2 in the form of ATM-text for phase 45 to process. Phase 4 sets a bit (PH45BIT) in the SWITCH1 cell in COMMON to indicate that phase 45 is to be called to process the UNSTRING verb, and passes the ATM-text to it. If phase 4 encounters a Q-Routine control break and the PH45BIT is on, it writes all Q-Routine text on SYSUT2.

The following sections give examples of phase 4 processing for several types of verbs. These examples show the general pattern of analysis for verbs and use most of the phase 4 tables.

#### MOVE STATEMENT -- SUBSCRIPTING

Phase 4 processing for the MOVE statement consists simply of producing a MOVE string that gives the number of operands and names the operands. For the input elements

MOVE A TO B

phase 4 generates the string

MOVE (2) A B

The operands of a MOVE statement may be subscripted. When subscripted operands are encountered in any statement, the generating routine first issues a SUBSCRIPT string for each subscripted operand and then issues the string for the verb. The following MOVE statement exemplifies the building of SUBSCRIPT strings:



STRING Table	DEFSBS Table	Output
MOVE (2)	SUBSCRIPT (3)	SUBSCRIPT (3) A 6 SSID1
SSID1	A	SUBSCRIPT (5) B C D E SSID2
SSID2	6	MOVE (2) SSID1 SSID2
	SSID1	
	SUBSCRIPT (5)	
	B	
	C	
	D	
	E	
	SSID2	

Figure 18. Tables and Output for the Statement MOVE A(6) TO B(C,D,E)

MOVE A(6) TO B(C,D,E) .

Processing for this statement is illustrated by Figure 18, which shows the contents of tables built for the statement and the P2-text strings produced.

**EXPLANATION:** The MOVE verb, with 2 to indicate the number of operands, is placed in the STRING table. Then the first subscripted operand is processed. For this operand, a SUBSCRIPT string is built in the DEFSBS table. (SUBSCRIPT is a special COBOL verb used only within the compiler; the DEFSBS table is a table similar to the STRING table, but it is used only to hold subscript information.)

The SUBSCRIPT string is used by phase 50 to resolve the subscripted reference. For the first SUBSCRIPT string shown in Figure 18, this string means "Compute the address of the sixth occurrence of A; place that address into a temporary cell called SSID1." At execution time, the address of the data item to be moved will be held in SSID1; therefore, SSID1 becomes the operand of the MOVE verb. The same applies to the second operand.

When the period ending the statement is encountered, all three strings are put out.

**DEBUG CARD**

If a procedure-name is referred to by a DEBUG card, phase 4 produces a CALL string.

The output generated for debugging procedures is illustrated in Figure 19.

When routine IDLHN analyzes a PN definition, it determines from the attributes that this PN is referred to on a DEBUG card. First, it issues a PN definition element. Then, it obtains a GN number from GNCTR in COMMON and issues a CALL string with this GN as its operand.

The PN number and GN number are saved together in table DBG TBL.

When a DEBUG card is encountered, the DBG TBL table is searched for a PN number that matches the one on the DEBUG card. In the table, this PN number has a corresponding GN number, and a GN definition element is issued for this GN. Therefore, the GN defines the location of the debugging procedure.

**ALTER STATEMENT**

For each ALTER statement in a program, two statements require ALTER processing: the ALTER statement itself, and the GO TO statement named in the ALTER statement. Either of these statements may be encountered first. When one statement of an ALTER/GO TO pair is encountered, a VN number is assigned, and a string of P2-text is written using this VN number. A VNTBL entry is made, giving the VN number and the PN number to which it corresponds. When the second statement of the pair is

Input Statements	DBG TBL Table	Output
PN1. ADD...	PN1 GN1	PN1. CALL GN1 ADD...
PN2. MOVE...	PN2 GN2	PN2. CALL GN2 MOVE...
DEBUG PN2		GN2. DEBUG PN2
DEBUG PN1		GN1. DEBUG PN1

Figure 19. DBG TBL Entries and P2-text for DEBUG Card Processing

encountered, this VNTBL entry supplies the VN number for this statement's P2-text output.

At execution time, each PN is assigned a cell in the Program Global Table. (For the format of the PGT, see the chapter "Object Module.") In this cell, the address of the first instruction for the PN is permanently stored. Each VN is assigned a cell in the VN field of the Task Global Table. (For the format of the TGT, see "Appendix B. Object Module.") However, the contents of these cells are not permanent. When a branch instruction is modified by an ALTER statement (or a PERFORM statement as described later in this chapter), the address contained in the VN cell is changed.

Figure 20 gives an example of phase 4 processing for two ALTER statements.

- 1 In this example, the first statement read is PN1. This is a GO TO statement referred to by an ALTER (the ALTER statement follows PN3).
- 2 When routine IDLHN examines the definition of PN1, it determines from the attributes that this statement is an altered GO TO statement. Then the GO verb analyzer processes the statement.
- 3 It obtains a VN number, which it stores with PN1 in the VNTBL table, and puts out two strings.
- 4 The GO VN1 string indicates that, when this branch is executed, the branch address is to be picked up from a uniquely identified VN cell in the TGT.
- 5 EQUATE VN1 PN3 indicates that at execution time the initial contents of this VN cell is the address of PN3. Until the value is changed by an ALTER statement, the cell will be unchanged,

and any execution of PN1 will branch to PN3. The equated address is also placed in a VN cell in the PGT.

(In a segmented program, the VN is given the same priority as the PN to which it is equated.)

- 2 When the ALTER PN1 statement is read, the VNTBL table is searched for PN1. Since an entry is found, the corresponding VN number (VN1) is used as the receiving field of the MOVE.
- 6 At execution time, this MOVE statement takes the address of PN2, which is stored in a PN cell in the PGT, and places it in the VN cell for VN1.

The execution-time operation of this ALTER/GO TO pair of statements is illustrated in Figure 21. The flow of control resulting from this ALTER statement is shown in Figure 22.

Figure 20 also illustrates a second ALTER/GO TO pair.

- 7 In this case, the ALTER statement (ALTER PN4) is read first.
- 8 A search of the VNTBL table reveals that no entry for PN4 has been made, so the ALTER analyzer obtains a VN number and enters that VN number with PN4 in the table.

Special Processing for Optimization: When the OPT option is in effect, most PN cells are eliminated from the Program Global Table. For addressing PNs without address constants in the PGT, phases 62, 63, and 64 use Procedure Block base locators. Some GN and PN cells remain unchanged by phases 62, 63, and 64. Phase 4 generates P2-text Optimization information elements (changed to Optimization A-text elements by phase 50) to identify the type of element that follows.

Input Statements	VNTBL Table	Output
1 PN1. GO TO PN3.	3 PN1 VN1	4 PN1. GO VN1
7 ALTER PN4 TO PROCEED TO PN6.	8 PN4 VN2	EQUATE VN1 PN3 MOVE PN6 VN2
PN2..... GO TO PN1.		PN2..... GO TO PN1
PN3..... 2 ALTER PN1 TO PROCEED TO PN2.		PN3..... 6 MOVE PN2 VN1
PN4. GO TO PN5.		PN4. GO VN2 EQUATE VN2 PN5
PN5.....		PN5.....
PN6.....		PN6.....

Figure 20. Table Entries and Output for ALTER Statements

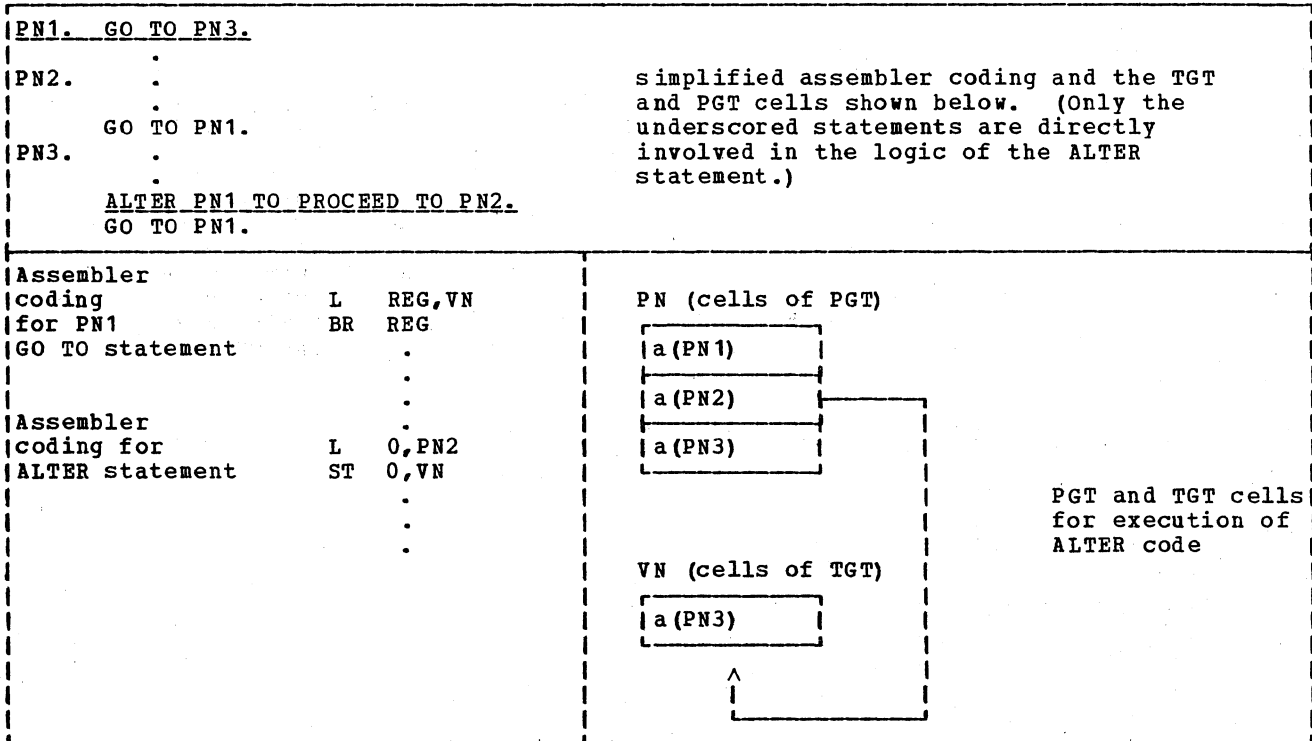


Figure 21. Execution of an ALTER Statement

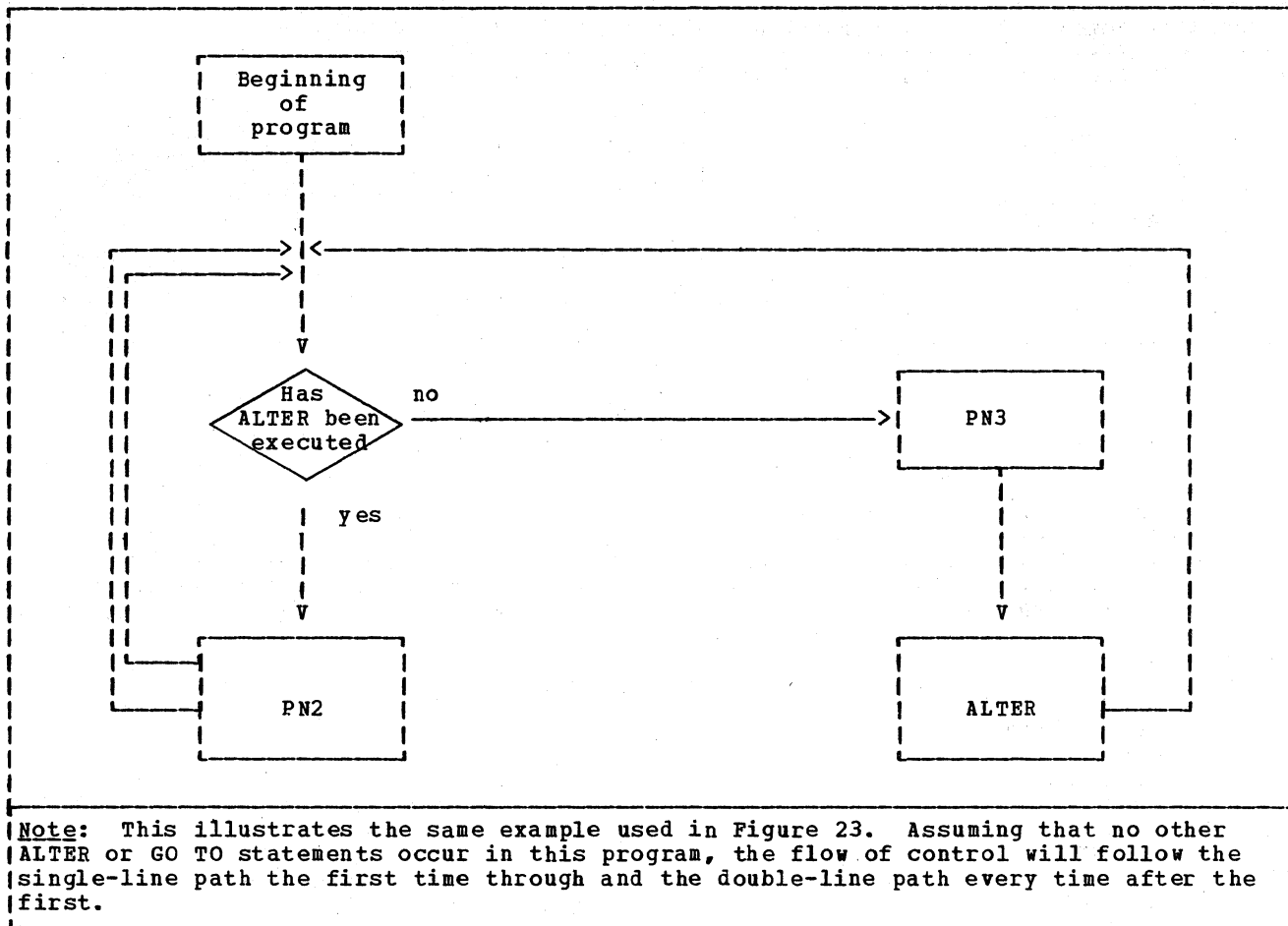


Figure 22. Flow of Control for ALTER/GO TO Statements

Procedure Statement	VNTBL Table	PFMTBL Table	Output
SN1 SECTION. ... SN2 SECTION. ... 1 PN3. ...	SN2 VN1 2	SN4 VN1 3	SN1. ... SN2. ... PN3. ... GO VN1 5 EQUATE VN1 SN4 6
SN4 SECTION. ... 8 PN5. PERFORM SN1 THRU SN2.			SN4. ... 7 PN5. MOVE VN1 PFMSAV1 9 MOVE GN1 VN1 GO SN1
PN6. ADD...			GN1. MOVE PFMSAV1 VN1 PN6. ADD...

Figure 23. Effect of a PERFORM Statement

PERFORM STATEMENT

Processing of a PERFORM statement resembles that of an ALTER statement. The return from a performed procedure is a GO string with a VN as its object. This GO string is placed at the end of the performed procedure, just before the procedure delimiter.

Phase 4 uses the VNTBL and PFMTBL tables to keep track of the VNs. Figure 23 gives an example of the use of these tables.

- 1 The dictionary attributes of section-name SN2 indicate that it is the object of the THRU option of a PERFORM statement.
- 2 Routine IDLHN obtains a VN number from the VNCTR cell of COMMON and enters VN1 and SN2 in the VNTBL table.
- 3 In the PFMTBL table, it enters VN1 and the delimiter of SN2, which is SN4.
- 4 When section-name SN4 is encountered, routine IDLHN knows it is the delimiter of the performed procedure because it is in the PFMTBL table. Therefore, before the procedure-name definition element for SN4 is issued, routine IDLHN sets up the return from the performed procedure.
- 5 It obtains the VN number from the PFMTBL entry.
- 6 It issues a GO string to go to VN1.

- 7 It issues an EQUATE string to equate VN1 to SN4.
- 8 When the PERFORM statement (PN5) is encountered, the verb analyzer PRFORM issues instructions to set up the return from the performed procedure.
- 9 It obtains a PFMSAV number from the PFMCTR cell of COMMON. This number represents a 4-byte cell in the Task Global Table of the object program. The value of VN1, which is the address of the next sequential instruction after the performed procedure, is saved by moving it into this PFMSAV cell. A GN number is obtained, and this GN is moved into the VN1 cell. This GN is defined at the point of return from the PERFORM. The instructions issued at this point cause restoration of the original value of VN1.

At execution time, sections SN1 and SN2 are first executed in-line. VN1 contains the address of section SN4, and thus the "return" from the procedure (the statement "GO VN1") actually causes control to pass to SN4, the next sequential instruction. Later in the program, the execution of statement PN5 causes SN1 and SN2 to be executed again. This time, the return is to GN1. The original value of VN1 is restored, and PN6 is executed.

Figure 24 gives an example of how a PERFORM statement operates at execution time. Figure 25 illustrates the flow of control for the program shown in Figure 24.

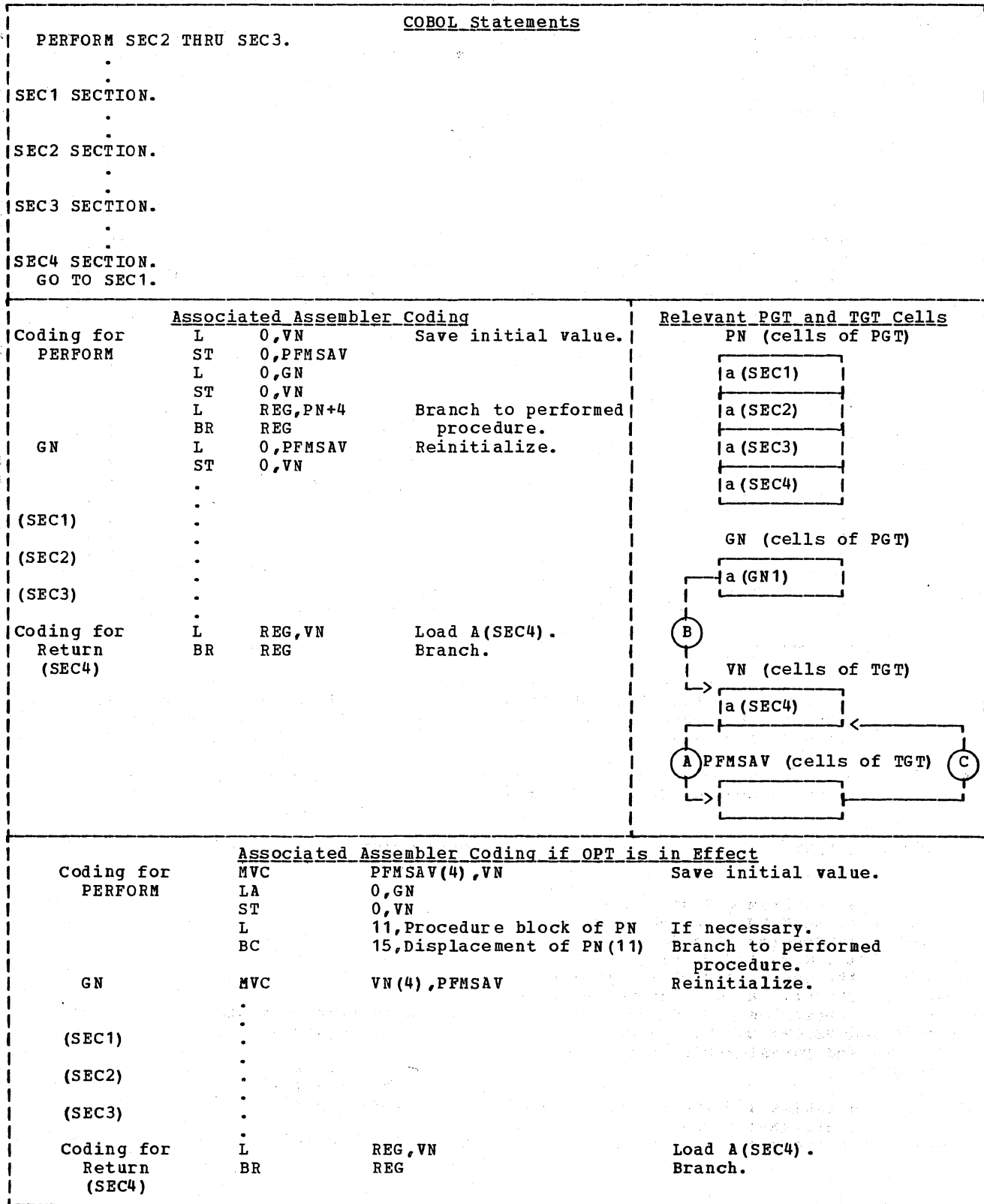
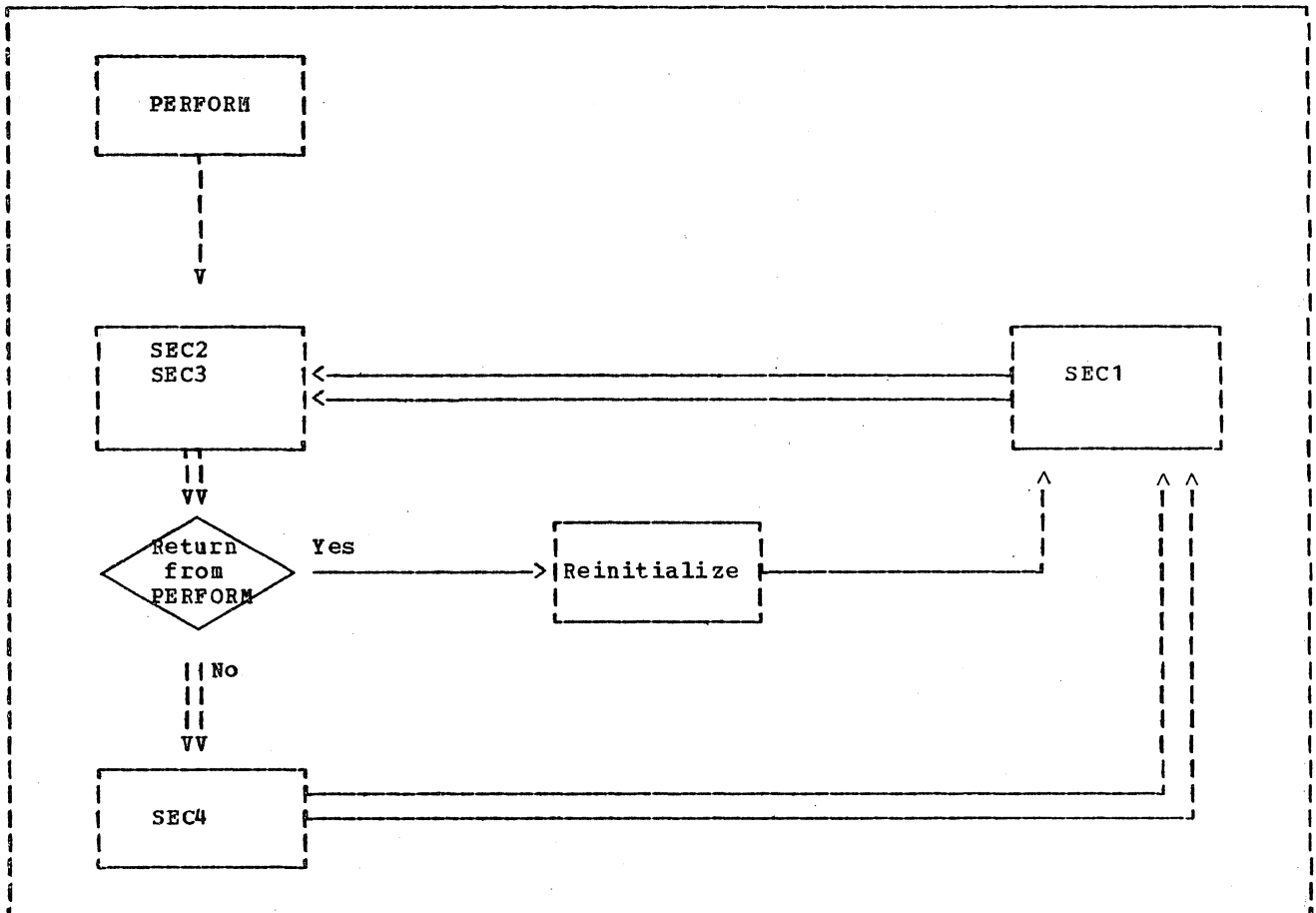


Figure 24. Execution of a PERFORM Statement



Note: This illustrates the same PERFORM statement shown in Figure 26. Assuming that no other procedure branching statements occur in this program, the flow of control will follow the single-line path the first time through and the double-line path every time after the first.

Figure 25. Flow of Control for a PERFORM Statement

COMPUTE STATEMENT

Verb analyzer routine COMPUTE, with its major subroutine FORMLA, breaks down the arithmetic expression of a COMPUTE statement into a series of simple arithmetic strings. It uses two tables, PNOUNT and PSIGNT, in the processing of the arithmetic expression. PNOUNT contains operands and PSIGNT contains signs (operators and parentheses) of the expression.

These two tables are needed because the hierarchy of arithmetic operators may necessitate a rearrangement of the

expression. The hierarchy in descending order is:

- unary -
- \*\*
- \* and /
- + and -

For example, the statement COMPUTE X = A-B\*C requires strings in the order

```
MULT C B IR1
SUB IR1 A IR2
STORE IR2 X
```

where IR1 and IR2 are intermediate results.

Input Element	PNOUNT Table Contents	PSIGNT Table Contents	Strings Stored in STRING Table
A	A		
+	A	+	
(	A	(	
C	A C	+ (	
-	A C	+ ( -	
D	A C D	+ ( -	
/	A C D	+ ( - /	
E	A C D E	+ ( - /	
)	A C IR1	+ ( -	DIV E D IR1
①	A IR2	+ -	SUB IR1 C IR2
*	A IR2	+ *	
F	A IR2 F	+ *	
-	A IR3	+ *	MULT F IR2 IR3
②	IR4	-	ADD IR3 A IR4
G	IR4 G	-	
③			SUB G IR4 IR5 STORE IR5 X

Note: The circled numbers in the figure refer to explanations in the text.

Figure 26. Evaluation of a COMPUTE Statement

Phase 4 builds a simple arithmetic string to be placed in the STRING table when the signs in the PSIGNT table indicate that the arithmetic hierarchy of operators requires a string. To build a string, the last operator in the PSIGNT table is made into a verb, the last two nouns in the PNOUNT table are used as operands, and an intermediate result is appended. (The intermediate result is placed in the PNOUNT table as an operand.) The following rules apply when a string is to be built:

- If there are no more input elements, all remaining strings are built. (see ③ in Figure 26)
- If a right parenthesis is encountered, all strings up to the left parenthesis preceding backwards into the tables are built. (see ① in Figure 26)
- If an operator is encountered that, in the hierarchy of arithmetic operators, is lower than or equal to the last sign in PSIGNT, a string is built. (see ② in Figure 26)

Figure 26 shows the evaluation of the following COMPUTE statement:

$$\text{COMPUTE X} = \text{A} + (\text{C} - \text{D} / \text{E}) * \text{F} - \text{G}.$$

Each row in the figure shows table contents after processing the input element in the first column.

Figure 27 gives the final output from the COMPUTE statement example used in Figure 26:

$$\text{COMPUTE X} = \text{A} + (\text{C} - \text{D} / \text{E}) * \text{F} - \text{G}.$$

EVAL DMAX DCURRENT X ...
DIV E D IR1 (-C)
SUB IR1 C IR2 (*F)
MULT F IR2 IR3 (+A)
ADD IR3 A IR4 (-G)
SUB G IR4 IR5 (ST X)
STORE IR5 X

Figure 27. Strings Resulting from a COMPUTE Statement



As an aid to phase 50, an EVAL string is issued preceding the arithmetic strings. The EVAL string contains information such as the maximum number of decimal places in any operand, the number of decimal places in the result, and the presence of ROUNDED or ON SIZE ERROR clauses. Appended to any string containing an intermediate result is an indication of the use of that intermediate result. For example, IR1 is used in a subtraction with C.

**MULTIPLE RESULTS IN ARITHMETIC STATEMENTS**

For COMPUTE with multiple operands before the equal (=), and for ADD, SUBTRACT, MULTIPLY, and DIVIDE with multiple GIVING operands, SETTBL is used to save the result operands. After determining whether SIZE ERROR is present, and analyzing the arithmetic expression for COMPUTE, SETTBL is used as the input stream to produce the store into each result. The routine STORAN is called once for each result.

**IF STATEMENT**

The IF verb analyzer, assisted by its major subroutine PFINDL, evaluates IF statements and issues strings consisting of a relational verb, two operands to be compared, and a third operand, which is a GN reference for branching. The following example shows the verb string issued from a simple IF statement:

<u>Statements</u>	<u>Strings</u>
IF A=B DISPLAY C.	IF-NOTEQ A B GN1 DISPLAY C
ADD...	GN1. ADD...

Note that the condition is reversed in the string to minimize the number of branches required. The branch to the GN is taken if the condition (A=B) is false, that is, if the DISPLAY is not to be executed.

The PNOUNT and PSIGNT tables are used in evaluating arithmetic expressions in IF statements. The strings produced are the same as for COMPUTE statements except that the last string is a relational string (for example, IF-EQ or IF-NOTGT) instead of a STORE string.

The PSHTBL and PTRFLS tables are used in evaluating IF statements. The PSHTBL table collects branches to ELSE statements in nested IF statements, and the PTRFLS table collects branches within compound IF statements.

Nested IF Statement

Figure 28 shows how the PSHTBL table is used in evaluating the following statement:

```
IF A=B THEN IF C=D THEN IF E=F STOP '0'
ELSE STOP '3' ELSE STOP '2' ELSE STOP '1'
ADD...
```

Procedure Statement	Status of PSHTBL Table	Output String
IF A=B	GN1	IF-NOTEQ A B GN1
IF C=D	GN1 GN2	IF-NOTEQ C D GN2
IF E=F	GN1 GN2 GN3	IF-NOTEQ E F GN3
STOP '0'	GN1 GN2 GN3	STOP '0' GO GN4
ELSE STOP '3'	GN1 GN2	GN3. STOP '3' GO GN4.
ELSE STOP '2'	GN1	GN2. STOP '2' GO GN4
ELSE STOP '1'		GN1. STOP '1'
ADD ...		GN4. ADD...

Figure 28. Evaluation of a Nested IF Statement

(IFs and ELSEs are paired from the inside outward.) The PSHTBL table saves the procedure-names (GNs) generated for branching to the ELSE statements. When an ELSE is encountered, the last GN in the table is issued as its procedure-name definition.

**SEARCH FORMAT-2 (SEARCH ALL) STATEMENT**

The SEARCH ALL statement is executed by a COBOL library subroutine. (For a description of the subroutine, see the publication IBM OS/VS COBOL Subroutine Library Program Logic. Therefore, phase 4 does not generate statements to perform the search. Instead, it produces a parameter list for the subroutine (the actual call to the subroutine is generated by phase 50),

and it creates verb strings for the imperative statements following AT END and WHEN clauses.

Figure 29 gives an example of phase 4 output for a SEARCH ALL statement. The example shows the P2-text that would be produced for the following statement:

```
SEARCH ALL TABLE-K AT END GO TO NOT-FOUND
WHEN KEY-1 (INDEX-K) = 5
AND KEY-2 (INDEX-K) = 10
AND KEY-3 (INDEX-K) = DATANAME-K3
MOVE TABLE-K (INDEX-K) TO ENTRY-FOUND.
```

- ① The verb analyzer first receives, as P1-text input, and saves TABLE-K and the maximum number of occurrences.
- ② Then it receives a count of the number of keys, followed by the keys themselves. The dictionary pointer for each key is entered into the KEYTBL table, with a flag byte of 00.

If any of the keys is found to be sterling or floating point, a C-level E-text element is generated, and the statement is processed as though it were a SEARCH format-1, with WHEN conditions being processed by the IF analyzer and no special WHEN statements generated.

After the table has been built, the analyzer transfers control to the SEARCH analyzer to scan the WHEN and AT END clauses.

To generate verb strings for the imperative statements following AT END and WHEN, the SEARCH verb analyzer calls other verb analyzers. However, the IF analyzer is not called to generate conditional statements. Instead, SEARCH returns control to the SEARCH format-2 analyzer to do special WHEN processing.

- ③ If the imperative statements following the AT END or WHEN clauses do not provide exits from the search, phase 4 generates GO TO NEXT SENTENCE. In the

example, this is accomplished via the GO GN4 statement.

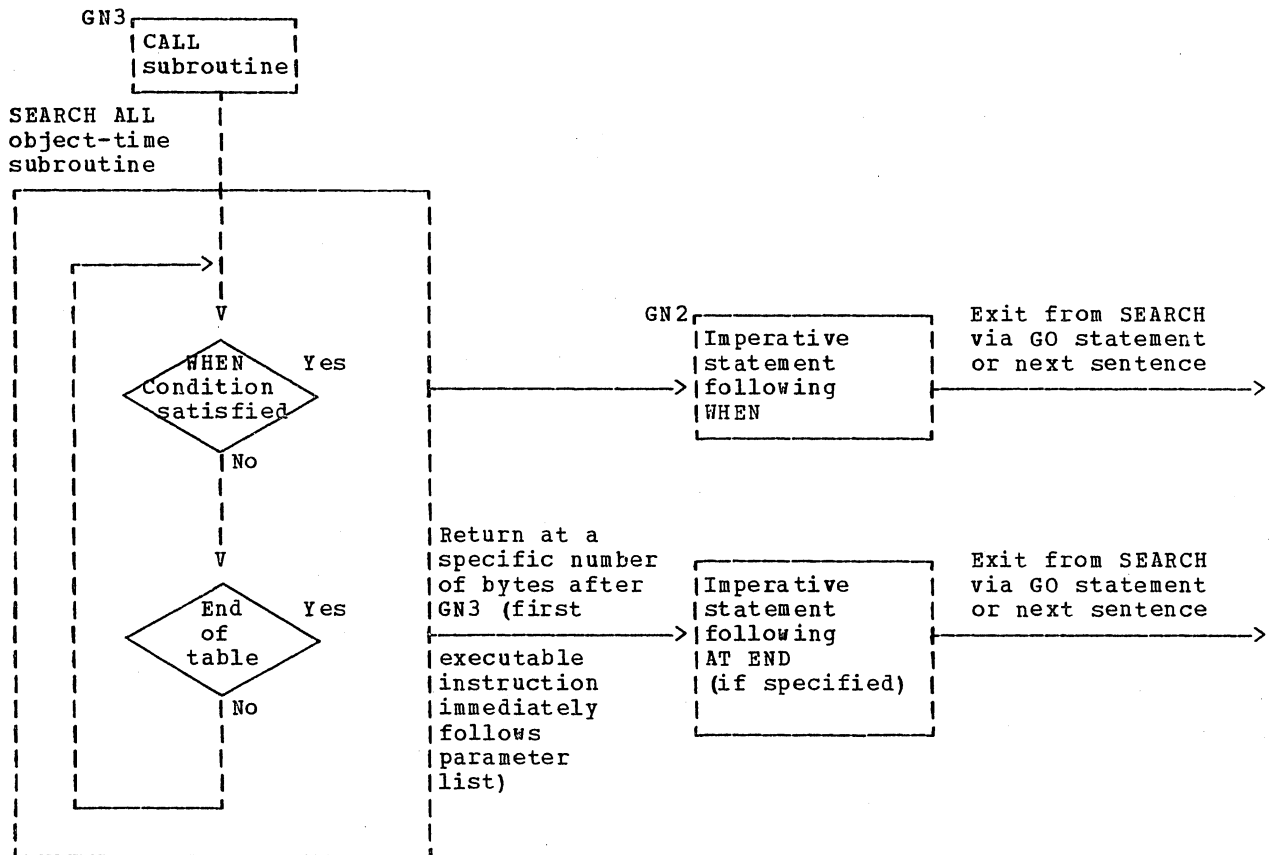
```
GO GN1.
GN2. GO PN50. (where PN50 is the PN
           number of NOT-FOUND)
GN3. WHEN (3) 1
    ① TABLE-K
      num-occur2
      GN2
    ② EVAL DMAX DCURRENT KEY-1.
      EQUATE 5 KEY-1.
      EVAL DMAX DCURRENT KEY-2.
      EQUATE 10 KEY-2.
      EVAL DMAX DCURRENT KEY-3.
      EQUATE DATANAME-K3 KEY-3.
      ENDWHEN (1) INDEX-K.3
      SUBSCRIPT (3) TABLE-K INDEX-K SSID1.
    ③ MOVE (2) SSID1 ENTRY-FOUND.
      GO GN4.
GN1. GO GN3.
GN4. next sentence
```

<sup>1</sup>If KEY-1 required two levels of subscripts or indexes, the first subscript would be put out as the fourth operand of the WHEN verb. If three levels were required, the first two subscripts would be operands 4 and 5 of WHEN. All keys must be subscripted or indexed when used in WHEN conditions, but phase 4 does not produce SUBSCRIPT strings or SSIDs for them.

<sup>2</sup>If TABLE-K has a fixed number of entries, num-occur is a literal specifying the number (the value following OCCURS in the data description for TABLE-K). If the table has a variable number of entries, num-occur is the name of the data-item which follows OCCURS... DEPENDING ON.

<sup>3</sup>The lowest level of indexing or subscripting for KEY-1 is passed as the operand of ENDWHEN, since it is needed for the search.

Figure 29. Example of Phase 4 Output for a SEARCH ALL Statement



(The largest box indicates control flow within the SEARCH ALL object-time subroutine; smaller blocks are executed in-line.)

Figure 30. Flow of Execution for a SEARCH ALL Statement

Figure 30 is a generalized chart showing the flow of execution into and out of the SEARCH ALL object-time subroutine.

Phase 4 checks WHEN conditions for conformity to language rules by using the KEYTBL table.

When the first part of the WHEN condition is processed, the dictionary pointer for the key named in the condition is compared to that of the first key in the KEYTBL table. If it matches, the flag byte is set to 01 and the condition is valid. Each part of the condition should name a key matching an entry in the KEYTBL table, and its flag byte will be set.

After the entire WHEN condition has been processed, the KEYTBL table is examined to determine whether any key is tested in the condition without all preceding keys being tested. If this occurs, E-text is issued and no P2-text is generated.

The user is not required to name the keys in the WHEN condition in any particular order. However, phase 4 must produce a P2-text string giving the keys in the order specified in the KEY IS clause. To produce text in the proper order, the STRING table is compared to the KEYTBL table. The STRING table entry that matches the first KEYTBL entry is used to produce the first text string, and the entries are then deleted from both tables. This process continues until all strings for the WHEN condition have been put out.

If, in any part of the WHEN condition, none of the operands named is a key, or if any other error condition is detected, E-text is issued and no P2-text is produced.

### SYNTAX ANALYSIS AND ERROR CHECKING

Phase 4, in subroutine ERROR, issues E-text when an error is detected.

Primarily, phase 4 checks to see that required COBOL words are present and in the correct order, and that operands are compatible with each other and in permissible format for the statement in which they are used.

In addition, phase 4 performs a limited check of the relationship between statements. For example, it can detect a conditional statement that has no conclusion.

It also checks miscellaneous special requirements, such as: subscripts must be integers; parentheses in logical and arithmetic expressions must be paired, and a maximum of twelve sort keys must not be exceeded.

If the CSYNTAX or SYNTAX option is in effect, E-text generated by phase 4 as well as the E-text passed from phase 3 is written on SYSUT4. Otherwise, E-text is written on SYSUT1. Phase 4 also sets the ERRSEV cell in COMMON to the highest error severity level encountered. If an error (E) or disaster (D) level message is generated and the CSYNTAX option is in effect, the SYNTAX option is forced into effect. If the SYNTAX option is in effect, phase 4 issues an end-of-job call if no messages were generated. If messages were generated, control is returned to phase 00 which sets the value of LINKCNT to indicate that phase 70 is to be executed next; error messages are listed and the compilation is terminated. (See "Syntax-checking Compilations" in the chapter "Phase 00.")

### METHOD OF DEFINING VERB BLOCKS

The major routines used in defining verb blocks and the functions they perform are explained below.

### Phase 40 Initialization Routine

This routine turns off the ENTRYSW, FNDPARNM, NNODEOS, NNODSW, NTIMSW, TIMFLOSW, UPPROC, and VBSKIP switches.

### ID5 Routine

Before branching to the verb analyzer routines, this routine calls TIMCNT.

### ISTRUV Routine

This routine turns on NNODSW and NNODEOS.

### IDBRK Routine

If not a report writer declarative, this routine turns on NNODSW, ENTRYSW, UPPROC, NTIMSW, and turns off NNODEOS and VBSKIP. If it is a report writer start, IDBRK turns on VBSKIP; if it is a report writer end, it turns off VBSKIP.

### IDLH03 Routine

Just before FLOW is tested, this routine turns on NNODSW, UPPROC, ENTRYSW, FNDPARNM, and turns off NNODEOS. Then, if FLOW is on, it turns on TIMFLOSW. Then GENTIM is called. In addition, if there is a DEBUG packet, at the end of IDLH10, GENTIM is called again. Note: the test and code generation for USE FOR DEBUGGING paragraph-name precede the tests for FLOW.

### SORT, MERGE Routines

Upon final exit from these routines, NNODSW is turned on if INPUT or OUTPUT procedures were specified. If INPUT or OUTPUT procedures are specified, NTIMSW is also turned on upon exit from processing the INPUT or OUTPUT procedure phrase.

### EOF Routine

If COUNT is on, this routine indicates the end of the COUNT Table in Data A-Text and rounds the size of the COUNT Table to the next even number.

GETNXT (GET13 and GET14) Routine

This routine saves Listing A-Text in order to maintain the last procedure-name for GENPAR.

EXIT5 (EXIT PROGRAM) Routine

Prior to generating this verb, this routine calls GENTIM, passing zero as the parameter to be generated. Afterwards, NTIMSW is turned on.

GENNOD Routine

If COUNT is off, or if VBSKIP is on, GENNOD returns. Otherwise, if NNODSW is on, NODECTR is updated. In any case, Data A-Text is generated. In addition, if NNODSW is on, a COUNT verb with a counter is generated, and NNODSW is turned off.

GENPAR Routine

First GENPAR turns off UPPROC. Then, if COUNT is off GENPAR returns. In any case,

Data A-Text for the paragraph, section-name, or missing paragraph-name is generated (using FNDPARNM), and FNDPARNM is turned off.

GENTIM Routine

First GENTIM turns off NTIMSW. If UPPROC is on, it calls GENPAR. In any case, if ENTRYSW is off, GENTIM generates a TIMERA verb (with a zero or PROCCTR as the count, depending on input parameters) and then returns. GENTIM turns off ENTRYSW, and if TIMFLOSW off, generates a TIMERB verb (with PROCCTR as the count) and returns. Otherwise it turns off TIMFLOSW and generates a TIMERC verb with PROCCTR as the first constant and with ISN or Card-number as the second constant.

WRSYS4 Routine

This routine puts the Data A-Text on SYS004.

PHASE 45

The function of phase 45 (IKFCBL45) is to produce P2-text on SYSUT1 for the UNSTRING verb. The phase is given control only if phase 4 encounters a valid UNSTRING verb string, in which case, phase 4 sets the PH45BIT switch in SWITCH1 in COMMON. Phase 45 is called by phase 00 following phase 4 processing if phase 00 finds the PH45BIT switch on.

Phase 45 operations consist of reading and analyzing the ATM-text for the UNSTRING verb, which phase 4 has written on SYSUT2, and transforming it into P2-text. The ATM-text consists of a series of subscript strings followed by an UNSTRING verb string; one or more of these series may be present.

INITIALIZATION AND ATM-TEXT ANALYSIS

Phase 45 first primes the four tables, SSCIN, SDELIM, SSCOUT, and TXTOUT, which are used during ATM-text analysis. These tables are used by phase 45 only.

After priming the tables, phase 45 begins to process the ATM-text from SYSUT2. The PH45CTL control routine examines the input verb string and gets the count of the operands in the string. It moves the operands into work areas labeled DOP1 through DOP5 (hereafter, a reference to a DOP is a reference to one of these work areas). The PH45CTL routine then checks whether the string is a subscript string. If it is, the string is saved in the SSCIN table and another input verb string is read. If it is an UNSTRING verb string, PH45CTL branches to the UNSTRING routine to process it.

The UNSTRING routine scans the operands and sets the appropriate flags to indicate operand type. It also checks whether the

sending field is subscripted. If it is, UNSTRING calls the FINDSSC routine to find the corresponding subscript string and enter it in the SSCOUT table. The FINDSSC routine also enters the data item text element in the TXTOUT table. If the sending field is not subscripted, the UNSTRING routine enters the data item text element in the TXTOUT table only. The UNSTRING routine branches to the proper field analyzer to process delimiter fields, receiving fields, delimiter-in fields, count-in fields, pointer fields, and tallying fields. The individual field analyzers set flags and make entries in the appropriate tables. When all of the DOPs have been processed, control is returned to the PH45CTL routine to refill the DOPs.

CREATING P2-TEXT FOR PHASES 50 AND 51

As ATM-text is analyzed and rearranged, P2-text is created for phases 50 and 51. The P2-text is generated so that the subscript for a subscripted field is calculated immediately before the field is referred to. Also, any Q-routines which must be called following the change of the object of an OCCURS...DEPENDING ON clause, are called immediately instead of after the entire verb string has been analyzed. P2-text elements, called Data-name information for UNSTRING (2A), are also created to pass information about UNSTRING data items, such as address and size, which is used by phase 51 when it generates Procedure A-text.

The P2-text is formatted in the SSCOUT and TXTOUT tables. The SORTXT routine sorts the corresponding sections of these tables, putting the text in the correct order. Finally, P2-text is written on SYSUT1.

Phase 50 (IKFCBL50) reads elements of P2-text written by phases 4 and 45 and, depending upon the type of each element, either processes it or passes it to phase 51 for processing. Output from phase 50 includes P2-text passed unchanged, Intermediate A-text, and Intermediate E-text, all written on SYSUT2 for phase 51, and Optimization A-text written on SYSUT2 for phase 6 or 62. Intermediate E-text consists of E-text passed from phase 4 or generated by phase 50, to which is added a prefix to make it readily recognizable. Intermediate A-text consists of Procedure A-text or Optimization A-text, which is generated by phase 50 and to which has been added a prefix. If the SYNTAX or CSYNTAX option is in effect, no prefix is added to Intermediate A-text.

Procedure A-text is generated by analyzing P2-text verb strings and generating elements that correspond to assembler-language instructions. That is, the elements combine to form name, operation, and operand fields. Optimization A-text is generated for use by phase 6 or 62 in eliminating duplicate references.

Input to phase 50 (a combination of P2-text and E-text) is read from SYSUT1. Output is written on SYSUT2, except for literals written as Optimization A-text on SYSUT3 (see the section "Literals and Virtuals" later in this chapter). In addition, if the SYNTAX or CSYNTAX option is in effect, E-text is written on a separate data set, SYSUT4; and the ERRSEV cell in COMMON is set to the highest error severity level encountered. (E-text processing is discussed under "E-text" in the chapter "Phase 51.")

Routine PH5CTL calls GETNXT, which obtains P2-text elements, determines what type they are, and calls the routines required to process them. The elements may be of the following types:

- Program breaks
- Verb strings
- E-text
- Segmentation control breaks
- PN, GN, and VN definitions

Of these, only program breaks and certain verb strings are processed by phase

50. The others are passed to phase 51 for processing. Elements of E-text are prefixed with headers for identification by phase 51, and copied as output. Segmentation control breaks and PN, GN, and VN definitions are written with an Optimization A-text prefix. Before a PN, GN, or VN definition is copied, all entries in the XSCRPT table, used in calculating subscript values, are deleted. The section "Using and Optimizing Subscript References" explains why this is necessary.

#### PROGRAM BREAKS

Routine GETNXT uses program breaks to determine where to issue a START macro-type instruction. This indicates where in the object-time coding the first executable instruction is generated. The Procedure Division, beginning-of-declaratives, and end-of-declaratives breaks are used for this. (All other types of program breaks are ignored.)

If no beginning-of-declaratives break is encountered, routine GETNXT issues the Procedure A-text for START immediately after finding the Procedure Division break. If a beginning-of-declaratives break is encountered, the START Procedure A-text is issued after the end-of-declaratives break has been found.

Since the coding for a START macro is always the same, this A-text is generated from Constant A-text, which is described later in this chapter.

#### VERB STRINGS

The P2-text for a verb string consists of a verb followed by from one to five operands. Any operands beyond the fifth have been placed into one or more continuation strings by phase 4; the first operand of the first string is the COBOL word FIRST, and the last element of the last string is the COBOL word END. (An example of this appears in the discussion of the DISPLAY string under "Verbs Requiring Calls to Object-Time Subroutines" in the chapter "Phase 51.")

Once routine PH5CTL has established that an element of P2-text is a verb string, it

moves the operands into work areas labeled DOP1 through DOP5 (hereafter, a reference to a DOP is a reference to one of these work areas). The verb code for the verb currently being processed is moved into a cell called GANLNO, where it is kept until overlaid by the next verb code.

The PH5CTL routine then calls routine XIS31, which determines whether any of the operands are subscripted or indexed data-names. If one is, the pointer to the desired occurrence has already been computed in phase 50 (subscripts and indexes are resolved by processing SUBSCRIPT verb strings, which always precede the verb string in which they are referenced; see the section "Resolving Subscripted and Indexed References" later in this chapter). The routine changes the idk (addressing parameters) field in the DOP from a subscripted or indexed reference to a data-name reference. (A full description of the idk field appears in "Dictionary Entry Formats" of "Section 5. Data Areas" under the addressing parameters field of the LD dictionary entry.)

Upon return, routine PH5CTL checks to see whether the verb string is one that is to be processed by phase 50. Verb strings processed by phase 50 include:

```

ADD
SUBTRACT
MULTIPLY
DIVIDE
EXPONENTIATION
Numeric IF
Numeric MOVE
SEARCH
DEBUG
INSPECT
DEBUG SUBSCRIPT
EVAL
STORE
SUBSCRIPT
EQUATE, when in SEARCH ALL
END OF, when in SEARCH ALL
    
```

The last five are verb strings created by phase 4. If the verb is one of those listed, an appropriate verb processor is called. Otherwise, routine PH5BVB is called, which performs some checking of phase 51 verb strings before passing them as P2-text to phase 51 (see "Handling Phase 51 Verb Strings" in this chapter).

**Note:** The PH5CTL routine distinguishes between numeric and nonnumeric IF and MOVE verb strings as follows: the numeric IF verb has a different verb code from the nonnumeric IF. For all MOVE strings, control is given to the numeric MOVE analyzer; if this routine finds nonnumeric operands, it returns control to routine

PH5CTL with an indication that such is the case.

#### VERB PROCESSING

From the verb code, routine PH5CTL determines which verb analyzer to call. If the STATE, TEST, or SYMDMP option is in effect, phase 50 generates a call to the object-time COBOL library debugging subroutine entry point ILBODBG4 before the code to call any other object-time subroutine except ILBOFLW1. The call to ILBODBG4 is generated only once per verb, even if the verb causes calls to more than one subroutine. For details on the object-time subroutines, see the publication IBM OS/VS COBOL Subroutine Library Program Logic. In general, there is a specific routine to analyze each COBOL verb, but there are a few cases of overlap. For example, all arithmetic verbs use parts of a processor known as the arithmetic translator.

The illustrations in this chapter show P2-text strings as a verb followed by data-names, and Procedure A-text elements as assembler-language instructions. These are simplifications for the reader: the texts actually contain codes rather than verbs, and the P2-text for data-names contains the dictionary attributes of the item, rather than its name. The actual text formats, including the codes, are shown in "Section 5. Data Areas."

**Note:** When phases 50 and 51 use registers 14 and 15 in their generated instructions, Procedure A-text DESTROY and RESERVE elements are passed to phase 6 or to phases 62, 63, and 64. The purpose of the DESTROY element is to indicate that the contents of the registers are not to be relied upon any longer. The purpose of the RESERVE element is to indicate that the register may not be used by phase 6 or by phases 62, 63, and 64 in the generated code until a FREE element for that register, issued by phase 50 or 51, is read.

#### RESOLVING SUBSCRIPTED AND INDEXED REFERENCES

This section describes the processing of the SUBSCRIPT verb string. It shows how subscripted addresses are calculated and how the XSSNT and XSCRPT tables are used to eliminate duplicate calculation. Then the handling of indexes (which are very similar to subscripts and use the same tables) is discussed.



How Subscripted Addresses Are Calculated

To refer to a subscripted item, the object program must know the displacement in bytes of the desired occurrence from the beginning of the subscripted field. To calculate this displacement, the following must be known:

- The number of bytes in the entire subscripted field.
- The relative position in the field of the desired occurrence.

The size of the field is known from the Data Division description of the field, including the PICTURE clause of the elementary item. The relative position is known from the value of the subscripts.

The examples which follow show three levels of subscripting. The same concepts (and the same formula) apply to one or two levels of subscripting.

The formula used to calculate a subscripted address is:

$$\begin{aligned} &(\text{subscript1} * \text{length1}) + (\text{subscript2} * \text{length2}) \\ &+ (\text{subscript3} * \text{length3}) \\ &- (\text{length1} + \text{length2} + \text{length3}) \end{aligned}$$

This formula is used whether the calculation is done in phase 50 or in the object program.

The following discussions use examples based on this entry in the Data Division:

- 02 FIELD OCCURS 10 TIMES.
- 03 SUBFIELD OCCURS 10 TIMES.
- 04 ITEM OCCURS 10 TIMES PICTURE XX.

Literal Subscripts: When all the subscripts in a reference are literals, the subscripted address can be calculated at compile time (the calculation is done by routine XSCOMP). If a reference is made to ITEM (9, 8, 7), the calculation is:

$$(9 * 200) + (8 * 20) + (7 * 2) - (200 + 20 + 2)$$

The value of the result is the displacement in bytes of ITEM (9, 8, 7) from the beginning of the subscripted field.

When the SUBSCRIPT string was first encountered, an entry was made for it in the XSCRPT table (the building of this table is more fully described in the section "Using and Optimizing Subscript References"). One field of the entry, called the idk field, contains a code, *i*, a displacement, *d*, and a base locator (BL) number, *k*. (BLs are assigned by phase 22; for a description, see the chapter "Phase

22.") The value of *d* is the displacement away from the BL of the beginning of the subscripted field. After the formula has been calculated, the results are added to *d*. If this value is less than 4096 bytes, it becomes the new value of *d*. If it is 4096 or greater, it is decremented by 4096; this decremented value is then placed in the *d* field of the XSCRPT table entry, and the BL number is incremented by one. The table entry now points to the desired occurrence at execution time. (Note that if the value of *d* exceeds a multiple of 4096, it is decremented by 4096 as many times as required to make it less than 4096, and 1 is added to the BL for each decrement. For example, if *d*=13,000, it is decremented by 12,288 and the BL number is incremented by 3.)

Data-name Subscripts: When the subscripts are data-names, their values vary at execution time. Therefore, code must be generated to perform the calculations in the object program. This is done in routine XSCOMP, using the A-text generator.

Suppose that reference is made to ITEM (X, Y, Z). XSCOMP must first determine whether each subscript is binary. If not, code must be generated to get the value of the subscript from the data area of the object program, move it into a work area, and convert the value in the work area to binary. It is then handled as a binary subscript, using the value in the work area rather than the data area.

The generated code performs the following actions at execution time:

- The value of *d* in the XSCRPT table (the address of the subscripted field) is loaded into a register.
- The first subscript (X in the example), in binary, is loaded into another register. This subscript denotes the desired occurrence number for FIELD, the highest level in the hierarchy. If FIELD is of constant length (that is, if it does not contain the DEPENDING ON option), the value of X in the register is multiplied by a literal representing the length of FIELD (200 bytes). If FIELD does contain the DEPENDING ON option, a value, instead of the literal, is picked up from a VLC (variable-length cell) where it was placed by a Q-Routine. This value represents the current length of FIELD.
- The result of this multiplication is added to the value of *d* in the first register, and this process is repeated for each subscript.

- Phase 50 has generated a literal (referred to in this discussion as LITX) whose value is:

(length of FIELD + length of SUBFIELD + length of ITEM).

- When the multiplications have been finished and the final increment has been added to the address in the register, this literal is subtracted from the register, and the result is the address of the desired occurrence.

This computation is an exact duplicate at execution time of the formula applied to literal subscripts at compile time.

After routine XSCOMP has put out the Procedure A-text for these instructions, it changes the idk field of the XSCRPT table entry for ITEM. It replaces the former contents with the number of the register which at execution time will contain the calculated address of ITEM (X, Y, Z), and a code indicating that this is a register number rather than a displacement.

#### Mixed Literal and Data-name Subscripts:

When the subscripts are mixed literals and data-names, for example, ITEM (X, 4, Z), part of the calculation can be done in phase 50 to save time in the object program. XSCOMP multiplies the literal subscript by the length of the field it refers to. In the example, this is 4\*20 (SUBFIELD is 20 bytes long). This result is subtracted from LITX before A-text for LITX is generated. Then, at object time, the multiplications and additions will be performed for X and Z and this new value of LITX will be subtracted.

Note that, to arrive at a meaningful value, the entire formula must be evaluated. The intermediate results of a subscript calculation are meaningless in themselves.

#### Using and Optimizing Subscript References

Part of the function of the XSCRPT table is to avoid duplicate calculations for a subscripted reference. For example, given the source statements:

```
PARAGRAPH1. MOVE ITEM (X, Y, Z) TO D.  
             ADD ITEM (X, Y, Z) TO E.
```

It is unnecessary to calculate the address of ITEM (X, Y, Z) twice. When the ADD statement is executed at object time, the correct address will already be in a register.

The P2-text for these statements would be:

```
PN1. SUBSCRIPT (5) ITEM X Y Z SSID1  
      MOVE (2) SSID1 D  
      SUBSCRIPT (5) ITEM X Y Z SSID2  
      ADD (2) SSID2 E
```

When the first SUBSCRIPT string is encountered, the XSCRPT table is searched for a matching entry. None is found, so an entry is made after the instructions for the subscript calculations have been generated. This entry includes the idk field described earlier and the dictionary pointers for ITEM and all the subscripts. (Note that the dictionary pointer is used only because it provides a unique identifier. The value of the pointer itself is meaningless.) There is also a field for the total length of the entry (the exact format can be found in "Section 5. Data Areas.")

An entry is also made in the XSSNT table. This entry includes the subscript number (SSID1) and a pointer to the XSCRPT entry.

After the SUBSCRIPT string has been processed, the MOVE string is encountered. Routine XIS31 (also called XSUDB3) searches the XSSNT table for SSID1. It uses the pointer in the XSSNT table to pick up the idk field in the XSCRPT table. This field replaces SSID1 in the DOP, and the verb is now processed as if the operand were a data-name.

The XSSNT entry for SSID1 is now deleted, since it will never again be referenced. This is because phase 4 gave a unique number to each subscript calculation.

When the second SUBSCRIPT string is encountered, the XSCRPT table is searched for a match. The search is made on total entry length (if this does not match, the entries cannot be identical). When the match is found, an XSSNT entry is made for SSID2, with a pointer to the same XSCRPT entry as for SSID1.

If there are not enough registers for the subscripted address to remain in a register for the second time it is referenced, code is generated before the register is used to store the address in a subscript save cell. (These cells are located in the SUBADR field of the Task Global Table.) An indication of this, along with the cell number, is placed in the XSNIDK field of the XSCRPT entry, so that the second time the address is needed, the contents of the cell can be loaded into another register. The section "Register and Storage Allocation" later in this

chapter describes how values in registers are saved.

Thus, the XSNIDK field may ultimately contain three types of information:

1. If all subscripts used in the subscripted reference were literals, the field contains a BL type, BL number, and displacement, in the idk format.
2. If one or more subscripts were data-names, the field has been updated to contain a register number and the code indicating that this is a register.
3. If the register is needed for some other purpose before the subscripted reference is used, the field contains the number of the subscript save cell in which the register contents have been stored, with the appropriate code.

Every time a PN, GN, or VN definition is encountered in the P2-text, the entire XSCRPT table must be deleted. The reason is shown by the following source program statements:

```

ON 2 AND EVERY 2 GO TO
PARAGRAPH2.
MOVE ITEM (X, Y, Z) TO D.
PARAGRAPH2. ADD ITEM (X, Y, Z) TO E.
    
```

In this example, if PARAGRAPH2 is entered through the branch in the ON statement, the subscript calculations generated for the MOVE will not have been executed, and the register will contain whatever value was left from the last time it was used. Therefore, the calculation must be performed in PARAGRAPH2. Deleting all the entries in the XSCRPT table whenever a paragraph-name is encountered assures that the code will be generated.

The XSCRPT table must also be deleted when any verb is encountered which can pass control to any point other than the next sequential instruction. An example of such a verb is a SORT verb or any VSAM verb.

### Indexed References

Indexed references are resolved by the same routines, applying the same logic, as subscripted references. The difference in their handling occurs because index-name values are never known at compile time, and because, at object time, the index-name cells (each index-name is a 1-word cell in the INDEX field of the TGT) contain values

expressing a displacement in bytes from the beginning of the indexed field. The value in the cell corresponds to the following formula:

$$(\text{occurrence number} - 1) * \text{length of elementary item}$$

It is the responsibility of the source programmer to set the value in the index-name via a SET statement before an indexed reference is made. The SET verb performs the same calculations for an index-name that routine SSCRPT does for a subscript.

Indexing may be direct, indirect, or mixed (indexes and literals). Direct indexing uses only index-names. Indirect indexing uses literals as increments or decrements. Mixed indexing consists of at least one index with one or more literal subscripts. Given the Data Division statements:

```

02 FIELD OCCURS 10 TIMES INDEXED BY A.
03 SUBFIELD OCCURS 10 TIMES INDEXED BY B.
04 ITEM OCCURS 10 TIMES INDEXED BY C
   PICTURE XX.
    
```

a reference to ITEM (A, B, C) would be direct indexing, and a reference to ITEM (A+4, B-5, C+6) would be indirect indexing.

Direct Indexing: The P2-text contains a SUBSCRIPT verb string with a special code in the operands to indicate that they are index-names rather than subscripts. Entries in the XSCRPT and XSSNT tables are made and used in the same way as for subscripted references. Object code is generated to place the value of d from the idk field into a register and add the values of all the index-names to it.

Indirect Indexing: Routine SSCRPT determines that there are literals in the SUBSCRIPT string. These literals are placed in an information gathering work area. When the XSCRPT entry is made for the indexed field referenced, the dictionary pointers are not entered, to prevent a match from being found. This is necessary for possibilities such as the following:

```

MOVE ITEM (A+4, B-5, C+6) TO D.
MOVE ITEM (A, B, C) TO E.
    
```

If a normal XSCRPT entry was made for the first statement, the dictionary pointers would be identical and the second statement would use the register set up by the first. Since the indexing does not point to the same place, the operation would be incorrect.

After routine XSCOMP has generated the code to add the index-names (as described

earlier under "Direct Indexing"), it tests the information-gathering work area. When literals are present, the routine generates additional instructions. These instructions load the literal into a free register, multiply the register by the length in the VLC for that level (in the example, the literal 4 would be multiplied by 200), and add this value to the register pointing to the indexed field (this register has already been incremented by the index-name values). This process is repeated for every literal. The multiplication must be done because the literals, unlike the index-names, represent occurrence numbers, not displacements in bytes.

Mixed Literal and Direct Indexing: Mixed indexing is processed as a combination of direct and indirect. All index-names are processed first, then each literal is treated as an indirect increment.

#### ARITHMETIC VERB STRINGS

The Arithmetic Translator is a group of verb analyzers used to process arithmetic verb strings. It is also used for all MOVE statements and IF statements processed by phase 50.

Given the source program statement:

```
COMPUTE X=A+(C-D/E)*F-G.
```

the following P2-text would be generated:

```
EVAL DMAX DCURRENT X...
DIV E D IR1 (-C)
SUB IR1 C IR2 (*F)
MULT F IR2 IR3 (+A)
ADD IR3 A IR4 (-G)
SUB G IR4 IR5 (ST X)
STORE IR5 X
```

For a description of how phase 4 generates this string, see "COMPUTE Statement" in the chapter "Phase 4" where this same example is used.

For each verb string, the Arithmetic Translator routines do the following:

1. Place information about each operand in a work area.
2. Determine the sizes of intermediate and temporary fields, and check for possible overflow by performing compile-time arithmetic.
3. Determine the mode for the operation.

4. Allocate registers and temporary storage for the operation.

5. Call the A-text Generator (described later in this chapter) to produce the Procedure A-text.

The formats of switches used by the Arithmetic Translator routines are given in Figure 31.

#### Work Area

For each operand in a string, a work area called an operand information buffer is set up. There are three types of operands:

1. Data-name operands. In the example, all the operands named in the source program COMPUTE statement are data-name operands.
2. Intermediate results. In the example, these include all the operands named with IR. Intermediate results are required because arithmetic machine instructions can handle only two operands at a time. The result of one arithmetic operation becomes an IR, which is then used as an operand of the next operation.
3. Temporary results. These are used by series addition and subtraction with multiple receiving fields. For the source statement:

```
ADD M, N, O TO P ROUNDED, Q
```

a temporary result is required to hold the sum of M, N, and O.

Each operand information buffer holds information similar to an entry in the table XINTR. This includes such attributes as the mode of the operand, the number of digits to the left and right of the decimal point, and the largest possible value of the operand. (The format of table XINTR is given in "Section 5. Data Areas.") For a data-name operand, these attributes are found in the P2-text element. For intermediate results or temporary results, the information is found in table XINTR, when the operand is used in the operation. It was placed in the table after processing of the string in which the intermediate or temporary result was created. The attributes for the intermediate result that is the result of the operation are filled in after processing the operands which form it (how the attributes are found is discussed in this section under "Compile-Time Arithmetic"). When the MULT string is processed, in the COMPUTE

XGSWT		XZSWT		
Bit	Meaning	Bit	Meaning	
0	Binary mode	0	XR3 contains address of second operand	
1	Arithmetic operation	1	Set XREG2 operand; XREG1 operand already set	
2	Internal decimal mode	2	Both operands processed	
3	Size error in floating-point conversion	3	Length preset	
4	Receiving field is floating-point	4	Use length after scaling	
5	End of ADD CORRESPONDING	5	Floating-point operation	
6	Size error option	6	Logical operation	
7	Rounded option	7	Binary	
XGNSW		Y1IRX+1 in Operand Buffers		
Bit	Meaning	Equate name	Bit	Meaning
0	Decimal place added for rounding	XLITZR	0	Zero literal
1	Divide rounded remainder option	XGNCON	1	Generated constant
2	IKP5011I-W already printed	XFTPT	2	Computational-1 or computational-2
3	Minus exponent	XLITER	3	Literal
4	Bypass store operation	XINREG	4	Operand is in a register
5	ADD CORRESPONDING	XGDPFP	5	Double precision floating-point
6	Size error	XGOVFL	6	Overflow may occur
7	Subroutine entered at entry point XOTSEM	XG2IRX	7	Double precision mode
XOPMOD		Flag Byte in Phase 50 Internal Table for Registers 0 through 5		
Bit	Meaning	Bit	Meaning	
0	All subscripts are numeric	0	Permanent flag for floating-point registers	
1	Subtract operation	1	Subscripts being computed	
2	Division operation	2	Register being used for subscript computation	
3	Divisor	3	Register being used for an intermediate result	
4	Multiply operation	4	Register being used in double precision mode	
5	Add or subtract operation	5	Register not available	
6	Floating-point mode	6	Register contains operand 2	
7	Unused	7	Register contains operand 1	

Figure 31. Arithmetic Processing Switches

statement example above, IR2 is found in the XINTR table. The attributes of IR3 are determined during processing of the MULT string, and IR3 is then used as an operand of the ADD string which follows.

If any of the operands are floating-point, the floating-point verb analyzer is called immediately to generate the Procedure A-text. Otherwise, compile-time arithmetic is performed.

**Note:** The operand information buffers are defined on the same storage as the data area for the SUBSCRIPT analyzer.

Compile-Time Arithmetic

Compile-time arithmetic is performed with the maximum possible values of the operands (9 in every digit position). If one of the operands is a literal, its actual value is used. The purpose of compile-time arithmetic is to determine if overflow is possible, and to determine how many places are required to hold the intermediate or temporary result.

For example, assume that for the string ADD IR3 A IR4 (-G), the attributes of A

show that it has a PICTURE of 99V9 and the attributes of IR3 (from table XINTR) show that it has a PICTURE of 9(3)V9(2). The compile-time arithmetic for the maximum value of the scaled operands is:

9990+99999=109989

The result determines the attributes (including the maximum possible value) of IR4, which are placed in table XINTR at the end of the ADD processing.

If the intermediate result (IR4 in this case) exceeds or is equal to  $10^{**}30$ , overflow is possible. To avoid overflow, instructions are generated to truncate the intermediate result down to 30 digits.

That is, let DMAX be the maximum number of decimal places in any operand in the source statement, let DIR be the number of decimal places in the intermediate result, and let IIR be the number of integer places in the intermediate result. Then the rules for truncation are:

1. If DIR exceeds DMAX and  $IIR+DMAX$  is less than or equal to 30, then low-order decimal places are truncated from DIR until  $IIR+DIR=30$ .
2. If DIR exceeds DMAX and  $IIR+DMAX$  is greater than 30, then low-order decimal places are truncated from DIR until it equals DMAX, and high-order integer places are truncated from IIR until  $DMAX+IIR=30$ .
3. If DIR is less than DMAX, then high-order integer places are truncated from IIR until  $DIR+IIR=30$ .

#### Mode of Operation

The mode of the operands determines the mode of operation. In general, the mode of operation is predetermined (for example, if one of the operands is floating-point, all of the operands are converted to floating-point and the operation is in floating-point). However, for operations involving binary and internal decimal operands, routine XHSMD is used to perform some tests to determine whether the operation is to be in binary or in internal decimal.

If any of the operands is external decimal, it is converted to internal decimal unless it is in a MOVE statement. If an operand is sterling nonreport, it is converted to internal decimal unless it is the target field of a MOVE or STORE verb.

If any conversions are required, they are handled by inline conversion or calls to COBOL library subroutines, depending on the complexity of the conversion. See the publication IBM OS/VS COBOL Subroutine Library Program Logic. for the generated calling sequences.

#### Register and Storage Allocation

To assign registers for an instruction, special register handling routines are used. These are:

- XRASG or XRSUAS to assign a single register
- XRDBIR to assign a register pair
- XRSASF to assign a floating-point register

Phase 50 maintains an internal table for registers 0 through 5, which are the arithmetic work registers of the object program. Each table entry contains a flag indicating whether the register is being used, how it is being used (for example, as one of a pair), and what it contains (such as a subscript or index calculation, or an intermediate result). The format of this flag byte is given in Figure 31.

If a register must be freed, routine XFREEER is called. The calling routine passes the number of the register to be freed to XFREEER in the XREGNO cell of the phase 50 data area. (XFREEER is also used by phase 50 when it is analyzing phase 51 verbs; see "Handling Phase 51 Verb Strings.") If registers 0 through 5 must be freed, routine DFREEER is called.

XFREEER checks the register table entry for the specified register and, if it is in use, generates instructions to store it. It must then indicate that the value formerly in the register is no longer there. If the register was being used for an arithmetic operation, it updates the operand information buffer by filling in the number of the TEMP STORAGE (object-time arithmetic temporary storage) cell in the TGT where the register contents are stored, and the displacement of the cell in bytes from the beginning of the TEMP STORAGE area. If the value in the register was a subscript, XFREEER changes the code of the idk (addressing parameters) field in the XSCRPT table entry (see "Using and Optimizing Subscript References" earlier in this chapter) to indicate that the value is in a subscript save cell, and provides the cell number. Subscript save cells are assigned in the SUBADR field of the Task

Global Table (TGT) during program execution.

Operations in binary are performed in registers, and the intermediate results are left there whenever possible. Decimal operations are performed in storage, with the intermediate results placed in cells of the TEMP STORAGE area.

Space in the TEMP STORAGE area is always allotted in 8-byte (doubleword) cells, regardless of how many bytes are actually required to hold the operand. (If more than eight bytes are needed, two cells are allotted.) To minimize total temporary storage area used by the program, each TEMP STORAGE cell is made available as soon as the value in it is no longer needed.

TEMP STORAGE cells for arithmetic operations are assigned by using the counter TSMAX in COMMON (for the format of COMMON, see "Section 5. Data Areas"). TSMAX contains the number of the highest numbered doubleword cell that has been assigned. When a new TEMP STORAGE cell is assigned, this counter is incremented by one. As soon as the value in the TEMP STORAGE cell is no longer needed, the cell number is placed in table XAVAL. The next time a TEMP STORAGE cell is required, XAVAL is searched first. If it has any entries, the cell found there is used, and the XAVAL entry is deleted. Only if table XAVAL is empty is a new TEMP STORAGE cell assigned from TSMAX.

Intermediate results are handled in such a way that, as soon as possible after an intermediate result has been computed, it is used in the next computation and is then no longer needed. In the COMPUTE statement shown above, for example, IR3 is the result of the MULT operation, and then is immediately used as an operand of the ADD.

Note that an intermediate result is also required after the last arithmetic operation (IR5 in the example) and that the value is then moved into the data-name result (X) via the STORE verb. This is necessary because the value may have to be converted, truncated, or aligned to conform to the format for X.

A temporary result, on the other hand, must be saved until all operations requiring it are finished. In the statement:

ADD M, N, O TO P ROUNDED, Q.

the temporary result (the sum of M, N, and O) is first added to P and then added to Q. All steps involving P (rounding, truncation, or conversion, if needed) are completed before the temporary result is

added to Q. When the temporary result is added to P, its value is moved into another cell of the TEMP STORAGE, and the operation is performed from that cell, leaving the original temporary result unchanged.

#### GENERATING A-TEXT

A-text is generated from three sources: Constant A-text, Direct A-text, and the A-text Generator. The nature of the text required determines which source is used to generate a particular block of text.

Constant A-text: This type of Procedure A-text resides in the phase 50 constant area. It is stored in the form of DCs, ready to be written out when needed. It is used for standard, frequently occurring object-time operations, such as START.

Direct A-text: This is generally written as one block of instructions at a time by routine GATXTV. The routine is called by the verb analyzers using two parameters:

- Displacement of the desired block of text from the beginning of the text area
- Length of text to be written

The verb analyzer fills in the variable fields before calling GATXTV.

A-text Generator: The A-text Generator is called by a verb analyzer to write one instruction of Procedure A-text at a time. It also generates Optimization A-text for virtual and literal definitions. It is used frequently by the Arithmetic Translator and by other analyzers requiring the generation of A-text too variable to be stored as Constant or Direct A-text.

To call the A-text Generator, the verb analyzer fills in the appropriate cells in the parameter area (see Figure 32) and calls a particular generating routine. The generating routine usually has the same name as the instruction it produces (for example, MVC, LOAD). The A-text Generator has no common entry point.

Before returning to the caller, the generating routine sets the entire parameter area to 0.

From the parameter information, the generating routine determines exactly what type of instruction to write. For example, if the LOAD routine finds that the only operands specified are two registers, it writes an LR instruction. If a nonregister operand is two bytes long, the generating

routine generates an LH. This occurs for a cell such as a VLC, which is always a halfword or a 2-byte literal. The A-text Generator does not test the length of a data-name.

Parameter Cells for the A-text Generator						
OPERAND-1 PARAMETERS			Defini- tion	Element	OPERAND-2 PARAMETERS	
Name	Meaning				Name	Meaning
OP1	Pointer to the DOP (storage cell in the phase) for a data-name operand.	1F	Address Reference	OP2	Same as for the first operand.	
XL1	Length of operand (used for SS instructions such as EX, MVC, AP). The routine, which generates the text, decrements this value by 1 before using it, as required by these instructions.	1H		XL2	Length of second operand (for SS instructions with two lengths, such as AP, SP, ZAP). The generating routine decrements the value by 1.	
XWC1	For arithmetic operands, specifies the operand's position in the TEMP STORAGE field of the TGT. TEMP STORAGE is used for arithmetic operands only.  <u>Bytes 1 and 2:</u> Displacement of the 8-byte slot containing this operand from the beginning of the TEMP STORAGE field.  <u>Byte 3:</u> Number of bytes actually required by the operation. TEMP STORAGE space is always assigned in slots of 8 bytes, but frequently an arithmetic operand is shorter, using only a few of the low-order bytes.	XL3		XWC2	Same as for the first operand.	
TALLY1	First operand is TALLY.	XL1	Address Reference	TALLY2	Second operand is TALLY.	
BDISP1	Base (specifies a register number) and displacement of operand.	XL2	Base and Displacement	BDISP2	Same as for the first operand.	

Figure 32. Parameter Cells for the A-Text Generator (Part 1 of 4)



Parameter Cells for the A-text Generator						
OPERAND-1 PARAMETERS			Defini- tion	Element	OPERAND-2 PARAMETERS	
Name	Meaning				Name	Meaning
RELAD1	Operand is referenced by its relative location within a field.  Byte 1: Code indicating the type of cell being referenced (the codes are listed under "Relative Address Element" in the Procedure A-text formats, given in "Section 5. Data Areas").  Byte 2: Number of bytes needed to express the address constant.  Bytes 3 and 4: Identifying number which pinpoints this cell within its field.	1F	Relative Address	RELAD2	Unused.	
VIRTC1	Identifying number assigned from VIRCTR when the operand is a virtual. The number begins in byte 3.	1F	Virtual Reference	VIRTC2	Unused.	
GVIRT1	Name of virtual (used with VIRTC1).	2F		GVIRT2	Unused.	
XCON1	Used for two distinct types of information:  (1) Operand is a literal  Byte 1-16: Value of literal, right-adjusted.  Byte 17: Length of literal.  (2) Operand is a DC-type constant other than an address constant  Byte 1: Length of constant.  Bytes 2-17: Value of constant, left-adjusted.	XL17	Literal Reference	XCON2	Same as for the first operand.	

Figure 32. Parameter Cells for the A-Text Generator (Part 2 of 4)

Parameter Cells for the A-text Generator						
OPERAND-1 PARAMETERS			Defini- tion	Element	OPERAND-2 PARAMETERS	
Name	Meaning				Name	Meaning
XGN1	GN number when operand is a GN reference.	1H	Generated Procedure-name Reference	XGN2	Same as for the first operand.	
XPN1	Operand is a PN reference.  Byte 1: Code 00  Byte 2: Priority number of PN  Bytes 3 and 4: PN number	1F	Procedure-name Reference	XPN2	Same as for the first operand.	
XCNTR1	Operand is an item in one of the variably-located fields of a Global Table.  Byte 1: Type code (the codes are listed under "Global Table Variably-located Area Reference" in the Procedure A-text formats, given in "Section 5. Data Areas").	XL3	Global Table Variably-located Area Reference	XCNTR2	Same as for the first operand.	
PLUS1	Displacement from beginning of allocated storage of a right-adjusted operand.	XL3		PLUS2	Same as for the first operand.	
XVN1	Operand is a VN reference  Byte 1: Code 00  Byte 2: Priority number  Bytes 3 and 4: VN number	1F	Variable Procedure-name Reference			
GDEBG1	Operand is an item in a fixed-location field of a Global Table.  Byte 1: Type code (the codes are listed under "Global Table Standard Area Reference" in the Procedure A-text formats, given in "Section 5. Data Areas").	XL2	Global Table Standard Area Reference	GDEBG2	Same as for the first operand.	

Figure 32. Parameter Cells for the A-Text Generator (Part 3 of 4)

Parameter Cells for the A-text Generator						
OPERAND-1 PARAMETERS			Defini- tion	Element	OPERAND-2 PARAMETERS	
Name	Meaning				Name	Meaning
BLREF1	Operand is a base locator.  Byte 1: Type of base locator: BL, BLL, SBL ("i" field of idk)  Byte 2: BL number	XL2	Base Locator Reference	BLREF2	Unused.	
XREG1	Register number if first operand is a register  Note: When this is used, the second operand (unless it is a register also) is still considered the first non-register operand and is placed in the operand-1 cell.	XL1		XREG2	Register number for second register in an RR instruction.	
IHM	Immediate field value for an SI instruction.	XL1				
XXREG	Register number for index register in an RX instruction.	XL1				

Figure 32. Parameter Cells for the A-Text Generator (Part 4 of 4)

LITERALS AND VIRTUALS

The A-text Generator does not include literals and virtuals in the Procedure A-text. Rather, it writes the virtual with an Optimization A-text prefix on SYSUT2 and writes the literal as Optimization A-text on SYSUT3. At execution time, virtuals and literals are stored in the Program Global Table. By processing Optimization A-text, phase 6 or 62 can eliminate duplicate storage if the same virtual or literal is used more than once.

For virtuals, the A-text Generator assigns an identifying number to the virtual and writes the number as Procedure A-text. The virtual itself is then written with an Optimization A-text prefix on SYSUT2, along with its identifying number.

Virtual identifying numbers are assigned from the VIRCTR cell of COMMON. This counter is initialized to 1 in phase 00. The counter is incremented to indicate the number of virtuals required by the options that are in effect. Each virtual and the options for which it is required are listed below. Although more than one option may

require the same virtual, the virtual appears only once.

<u>Virtual</u>	<u>Options that require the virtual</u>
SRV0	NORESIDENT
DBG0	COUNT, FLOW, STATE, SYMDMP, TEST
DBG5	SYMDMP
FLW0	FLOW
STN0	STATE
CT10	COUNT
TO00	COUNT
SGM0	TEST, segmentation

The total number of virtuals processed by phase 50 is equal to the number of virtuals required by the options in the above list. At the end of phase 50 initialization, the value in VIRCTR is one greater than the total number of virtuals processed by the phase. Within phase 50, whenever a new virtual is needed, the current value of VIRCTR is used as the virtual number, and then the counter is incremented. At the end of phase 51, the counter is decremented by 1. (VIRCTR is the only counter in COMMON handled in this manner. For all other counters, the value of the counter is incremented before being used.)

When a literal reference is required, the Procedure A-text element contains a code and a counter. The literal itself is put out as Optimization A-text, and the LTLCTR counter of COMMON is incremented. An identifying number is assigned to the literal, so that phase 6 or 64 can determine which literal reference applies to a given literal.

During phase 50 processing, the Optimization A-text for a literal definition is written in SYSUT3; all other output from phase 50 is written in SYSUT2.

#### HANDLING PHASE 51 VERB STRINGS

Once the PH5CTL routine has determined that a verb string is not one of those processed by phase 50, it calls routine PH5BVB to handle the verb string.

The PH5BVB routine first checks to see if the verb is one that will use registers 0 through 5 at execution time. If it is, routine DFREER is called to free registers. (DFREER is described in "Register and Storage Allocation" earlier in this chapter.)

Routine PH5BVB then writes the header and operands of the verb string on SYSUT2 as P2-text. Then it determines whether the verb is MOVE, EXAMINE, TRANSFORM, RECEIVE, or SEND; a MESSAGE condition; or a subroutine test verb generated for an IF (NOT) MESSAGE statement. If it is one of these, routine XSPRO is called. This routine checks to see whether there is an object of an OCCURS clause with the DEPENDING ON option in the data-name being moved into, examined, or transformed, in the CD-name, or in the data-name associated with a RECEIVE or SEND verb. When this is

the case, routine XSPRO generates calls to Q-Routines.

Finally, routine PH5BVB determines whether the verb is one that requires deletion of all the entries in the XSCRPT table. These verbs include:

CALL	WRITE	RECEIVE
LINK	REWRITE	ACCEPT MESSAGE
ENTRY	REPORT-CALL	CANCEL
SORT	RELEASE	VSAM READ
OPEN	ON	VSAM WRITE
CLOSE	STOP	VSAM REWRITE
ACCEPT	STRING	VSAM DELETE
RETURN	UNSTRING	VSAM START
READ	SEND	VSAM OPEN
MERGE		VSAM CLOSE

If the verb is one of the above, routine KILSUB is called to delete the XSCRPT table entries.

#### ADDITIONAL PROCESSING FOR THE OPT OPTION

If the OPT option is in effect, phase 50 also performs the following functions:

- Converts phase 4 Optimization Information elements (43xx) to phase 50 Optimization Information elements (C0xx).
- Primes the BLUSTBL table for the number of BLs and BLLs present in the program. For each reference to a data-name, it adds 1 to the entry for that BL or BLL. This table is used by phase 62 to assign permanent base registers to the BLs and BLLs. Phase 51 also updates the BLUSTBL table.
- Writes GNUREF elements for Q-Routine calls.

PHASE 51

Phase 51 (IKFCBL51) functions in a similar manner to phase 50. Elements of text written by phase 50 are read from SYSUT2. Phase 51 checks each element and performs whatever processing is required, based on the type of element read. After processing, it writes the element as Procedure A-text on SYSUT1, as Optimization A-text on SYSUT3, or as E-text on SYSUT4. (Optimization A-text is written immediately after any Optimization A-text that was written on SYSUT3 by phase 50.)

Input to phase 51 can be any of the following:

- Intermediate Procedure A-text
- Intermediate Optimization A-text
- Verb strings
- Intermediate E-text

An element of Intermediate Procedure A-text may be a Q-Routine control break, or it may be text generated by phase 50 and requiring no further processing. If it is a Q-Routine control break, routine GETNXT generates an end-of-file macro instruction, if necessary, and writes the text element of 4440. Otherwise, the element is merely copied as Procedure A-text output without the identifying prefix of 28 and its count field.

An element of Intermediate Optimization A-text, identified by a prefix of 27, can be a segmentation control break or a PN, GN, or VN definition.

Verb strings written in P2-text require processing by one of the phase 51 verb analyzer routines.

E-TEXT

Whenever it encounters E-text in its input or generates an E-text element itself, phase 50 writes it out on SYSUT2 with an identifying prefix or, if the SYNTAX or CSYNTAX option is in effect, on SYSUT4 without the prefix. E-text with the prefix is referred to as Intermediate E-text. Phase 51 recognizes it, discards the prefix, and writes the E-text out on SYSUT4.

To enable phase 00 to pass a return code back to the operating system at the end of compilation, phase 51 must determine the highest severity level encountered in the program. When routine GETNXT encounters an element of E-text or when the phase 51 processing routines find an error situation requiring that E-text be written, routine ERRPRO is called. This routine uses a cell in COMMON called ERRSEV (for the format of COMMON, see "Appendix A: Communications Area"). If any E-text was generated by phases 10, 12, 20, 22, and 21, ERRSEV was set by phase 21. Thereafter, ERRSEV is set by phases 3, 4, and 50 to the highest error severity level encountered.

Routine ERRPRO adds 1 to the severity code of the current E-text element and multiplies this value by 4. (The code must be incremented by 1 because certain errors produce a severity code of 0; adding 1 to the severity code distinguishes such an error from no errors at all.)

ERRPRO then compares this value to the current value of ERRSEV, and enters the code of the E-text into ERRSEV if it is higher. The E-text is then written out on SYSUT4. Note that this is the first time (unless the CSYNTAX or SYNTAX option is in effect) that E-text is separated from the other output of a phase, rather than embedded in it.

If the CSYNTAX option is in effect and if any phase generates an error (E) or disaster (D) level message, the SYNTAX option is forced into effect. Following phase 50 processing if phase 00 finds the SYNTAX option in effect, phase 00 sets the value of LINKCNT to indicate that phase 70 is to be executed next. If phase 00 finds that the CSYNTAX option is still in effect, processing continues with phase 51. If phase 51 detects no syntax error which would cause an error (E) or disaster (D) level message, a full compilation is produced. Otherwise, the SYNTAX option is forced into effect, the ERRSEV cell is set, and control is passed to phase 00. Finding the SYNTAX option in effect, phase 00 sets the value of LINKCNT to indicate that phase 70 is to be executed next; error messages are listed and the compilation is terminated (see "Syntax-checking Compilations" in the chapter "Phase 00").

## PROGRAM BREAKS

Routine GETNXT uses program breaks to determine where to issue the START macro-type instruction. This instruction indicates where in the object program control is to be passed by the initialization routines (see the discussion of these routines in the chapter "Phase 6") if the program is executed by PROGRAM-ID rather than at an alternate entry point.

Each program break read by phase 51 (except for segmentation control breaks that are handled as described below) is handled by the routines CHKSEG and STRTEST. CHKSEG examines the type of break and sets the bit configuration of STRTSW, a one-byte switch. STRTEST checks STRTSW to determine the exact point at which the START macro-type instruction should appear. STRTSW is also used to check the point at which current processing is taking place, for example, within the Report Writer Section.

If no beginning-of-declaratives break is encountered, the START is written immediately after the Procedure Division break is encountered. If declaratives are present, the START is written after the end-of-declaratives break has been found.

## SEGMENTATION CONTROL BREAKS

Phase 51 writes Procedure A-text on the direct-access device SYSUT1. When phase 6, or phases 62 and 63, read this text, segments must be read and processed in order of priority, rather than in the order in which they appear in the source program. To facilitate this, routine SEGENTR builds a table (called SEGTBL) containing the relative disk address of the beginning of each segment.

The relative disk addresses are obtained through phase 00. (Note that phase 00 handles all input/output requests for the other phases.) When phase 00 writes Procedure A-text for the first physical record of a segment, after the CHECK has been issued for that WRITE, it issues a NOTE macro instruction. The NOTE macro instruction returns the relative disk address of the record just written, which is saved in SEGSAVE, a cell internal to phase 00.

The segmentation control break encountered by phase 51 signals the end of one segment and the beginning of a segment with a different priority. Routine GETNXT calls phase 00 with a request for the

SEGNOTE function. (For a description of the calling sequence and parameters, see "Phase Input/Output Requests" in the chapter "Phase 00".) To complete the output for the previous segment, phase 00 writes whatever is left in the buffer, and passes back to phase 51 the value in SEGSAVE, which is the starting address of the previous segment. Phase 51 stores this as an entry in SEGTBL, along with the priority number of the previous segment. (For the SEGTBL table format, see "Section 5. Data Areas.")

The call to SEGNOTE also indicates to phase 00 that the next time it writes on SYSUT1, it will be the beginning of a new segment and another NOTE must be issued.

## PN, GN, AND VN DEFINITIONS

When a PN, GN, or VN definition is encountered, routine PUTDEF is called. This routine changes the definition to Procedure A-text and writes it out.

For PN definitions, the PN number and the priority number are saved before PUTDEF is called. For VN definitions, the SEGLMT cell in COMMON (see "Appendix A: Communications Area") is tested to see if the program is segmented (a value other than hexadecimal FF in SEGLMT means the program is segmented). If it is segmented, the VN definition is written as both Procedure A-text and Optimization A-text.

## Building PN and GN Equate Strings

It is possible that earlier phases have generated more than one GN to define a single verb within a procedure statement. When this occurs, several GN definition elements are encountered in a row without any intervening text. If the source program included a procedure-name at this point, a PN definition also occurs, preceded by one or more GN definitions. Since only one procedure-name is required to provide a branching point in the object program, the rest of the GNs can be eliminated. All the GNs (and the PN, if any) are collected in a phase 51 work area called GNLIST, from which they are written in Optimization A-text as an Equate string. In Procedure A-text, only the PN definition is written or, if there was none, the first GN definition (the one with the lowest GN number). Phase 6 uses the Equate string to change all references throughout the program from the equated GNs to the one for which Procedure A-text was issued.

If the OPT option is in effect, these PN and GN Equate strings are not written. All PNs and GNs are written in Procedure A-text regardless of their position. Since no cells in the PGT are allocated for their addresses, it is not necessary to equate their addresses. They are addressed instead by using a displacement from a base register.

verbs, it calls routine DBGTEST to generate the call. The call is also generated before the code generated to call any COBOL object-time subroutine or any program that is the operand of a CALL statement, and before any branch to a Q-Routine or to report writer routines (not including report writer declaratives). For a description of the COBOL library subroutines, see the publication IBM OS/VS COBOL Subroutine Library Program Logic.

### Building the PNTBL Table

The source programmer may have coded some procedure-name definitions to which reference is never made. The address cells for such procedure-names can be eliminated. To do this, routine PNUSED in phase 51 builds the PNTBL table for phase 6 or 62.

This table contains one bit for every PN definition in the program, rounded up to a multiple of 4. The size of the table is determined from the cell PNCTR in COHHON. This cell was incremented by phase 1B every time a PN definition was created, and therefore contains a count of the total number of PNs in the program. All bits in the table are initialized to 0, and every time a PN is referred to throughout phase 51 processing, the bit in the table corresponding to the PN number is set to 1. Phase 62 uses the PNTBL table to determine if a PN has been referenced. If it has not been referenced, no entry point processing is done at the point of definition.

### VERB STRINGS

Phase 51 processes input/output verbs, other nonarithmetic verb strings (including some requiring calls to object-time subroutines), and DISPLAY literals. As examples, the ON string is discussed below under "Other Nonarithmetic Verb Strings" and the DISPLAY verb under "Verbs Requiring Calls to Object-Time Subroutines." Samples of coding are included in those discussions.

If the STATE or the SYMDMP option is in effect, a call to the COBOL library debugging subroutine entry point ILBODBG4 is generated for all verb analyzers (except FLOW and COUNT) which produce code branching outside the main line of the program. (However, this call is not generated before the call to the flow trace subroutines entry point ILBOFLW1.). When routine PH5CTL encounters the READ, WRITE/REWRITE, OPEN/CLOSE, RETURN, RELEASE, USE/ENDUSE for error and label declaratives, and checkpoint READ and WRITE

### Input/Output Verbs

Phase 51 generates the object code required for the input/output verbs discussed in this section. There is a separate verb analyzer routine for each verb. Each routine may share several subroutines with other analyzers.

The coding generated is basically the required linkage to an input/output routine and, therefore, depends on the access method. Additionally, the following factors influence the coding: the organization of the data records, the blocking factor, the recording mode, multiple buffering, use of SAME RECORD AREA clause, inclusion of a RERUN clause, use of label records and declaratives, and the type of device.

By convention, the following register assignments are used at execution time.

- Before an input/output operation:

- If a DECB for a file is referred to in the generated code for an input/output operation, the address of the DECB is placed in both registers 1 and 3, and the address of the DCB for the file is placed in register 2.

- If a DECB for a file is not referred to in the generated code, the address of the DCB for the file is placed in both registers 1 and 2.

- Register 15 contains the address of the input/output routine that corresponds to the particular operation and access method.

- After certain input/output operations:

- Register 1 contains the address of the next record to be read or the next area to be written in.

- Before calls to object-time subroutines:

- Register 15 contains the address of the COBOL library subroutine to be called.
- Register 1 contains the address of the parameter list, if any.

**Note:** Before the first inline call for a SORT or MERGE statement to the ILBOSRTO subroutine, registers 0 through 5 contain parameters used by the subroutine.

The code generated for input/output verbs is described in the paragraphs that follow. In cases where a call to a COBOL library subroutine is generated, the calling sequence is given in the publication IBM OS/VS COBOL Subroutine Library Program Logic.

**OPEN1:** The general form of the code is the expansion of the OPEN macro. Several additions may be made depending on the device and access method used. A call to the ILBOSAM0 subroutine is generated for a BDAM file opened as output. Code is generated to move the address of the Error subroutine, ILBOSYNx, into the SYNAD address in the DCB, to move the address of ILBOSPA0 into the DCB extension, to move the address of ILBOEXT1 into the exit list, and if RERUN has been specified, to move the address of the Checkpoint subroutine, ILBOCKP0, into the exit list. The input is two strings, the first of which is used to generate device-type code and the OPEN macro expansion. The second is used to generate the device-dependent code needed after the file is open. In addition, code is generated for each file so that an appropriate message can be produced on the console in the event of an unsuccessful open for a file.

**CLOSE1:** The code generated is the expansion of the CLOSE macro. Additions may be made depending on the device and access method used. A call to the ILBOSAM0 subroutine is generated for the CLOSE statement for a BSAM file.

**READ1:** This verb may cause either of two distinct sets of code: a GET or READ macro expansion is generated in most cases, with additions depending on the device and access method used; a COBOL library subroutine linkage is generated in special cases.

-----  
<sup>1</sup>For VSAM files, all interfaces to the VSAM access method are handled by the library subroutines ILBOVOC and ILBOVIO. These subroutines are described in IBM OS/VS COBOL Subroutine Library Program Logic.

**WRITE1:** This verb may cause either of two distinct sets of code: a PUT or WRITE macro expansion is generated, with additions and modifications depending on the device and access method used; a COBOL library subroutine linkage is generated in special cases, for example, a WRITE statement for a BSAM file, or QSAM file, or printer spacing.

**Note:** For the OPEN, CLOSE, READ, and WRITE verbs, Procedure A-text BLCHNG elements are written. This indicates to phase 6 or to phases 62, 63, and 64 that if they have permanently or temporarily loaded the associated BL into a register, they must reload the register or flag the register as no longer containing the BL. For ENTRY, SORT, MERGE, label declaratives initialization, and referencing the TOTALED AREA, Procedure A-text BLCHNG elements followed by the BLL reference instead of the base locator (BL) reference are written to indicate that a BLL has changed and must be reloaded into the permanently assigned register or that the internal phase 62 temporary register cell (14 or 15) must be flagged as no longer containing the BLL.

**START1:** The START statement for ISAM files without the generic KEY specified, results in the macro expansion of an ESETL macro instruction, followed by a SETL macro instruction without the key class option.

The START statement for ISAM files with the generic KEY specified, results in the generated code for a call to the START subroutine. Following the subroutine call, phase 51 generates the macro expansion of the ESETL macro instruction, followed by a SETL macro instruction with the key class option.

**DISPLAY:** The DISPLAY statement results in the generated code for a call to the ILBODSP0 or ILBODSS0 subroutine except for a DISPLAY of an alphanumeric literal, which results in the macro expansion of a WTO macro instruction.

**ACCEPT:** The ACCEPT statement results in the generated code for a call to the ACCEPT subroutine, ILBOACP0, when the "FROM SYSIN" option is used. When the "FROM CONSOLE" option is specified or implied, a WTOR macro expansion is generated.

**RECEIVE:** The RECEIVE statement results in the generated code for a call to the RECEIVE subroutine, ILBOREC0. The ACCEPT MESSAGE statement also results in the generated code for a call to the RECEIVE subroutine, ILBOREC0. A switch byte is set to indicate that the call is for a message condition rather than a RECEIVE verb.



**SEND:** The SEND statement results in the generated code for a call to the SEND subroutine, ILBOSND0.

**STRING:** The STRING statement results in a call to the STRING subroutine, ILBOSTG0.

**UNSTRING:** The UNSTRING statement may result in one or more calls to the UNSTRING subroutine, ILBOUST0, and in two or more calls to the conversion subroutine, ILBOCVB0, at both entry points: ILBOCVB0 and ILBOCVB1. The number and order of calls are determined by the types of operands specified in the UNSTRING statement.

**USE:** The USE verb, on entry to the Declaratives Section, generates code which sets up fields (pointers) for the information requested, such as the address of a label or an error block. At the end of the section, the code needed to return to the object-time subroutine is generated.

Other Nonarithmetic Verb Strings

This section discusses the ON string as an example of a nonarithmetic verb string.

When routine PH5CTL encounters an ON string, it moves the operands into a work area and calls routine ON to process the string.

The processing depends upon the options given in the ON statement. In the simplest case (ON 1), instructions are generated to test a switch to see if the statement completing the ON has been executed and, if it has, to branch around this statement.

Figure 33 shows the Procedure A-text produced for an ON statement with an initial value, increment, and maximum value. The numbers of the following explanations refer to the circled numbers in the figure.

- ① GN1 is the generated procedure-name assigned to the next sequential source program statement. A branch must be made to this statement when the ON condition is false (that is, in this example, when the ON instructions to test and increment the counter have been executed an odd number of times or more than 16 times).
- ② This instruction loads the contents of ONCTR1 into register 3. ONCTR1 is the identifying number of an ON control cell. At execution time, this cell will be incremented by 1 each time the ON statement is executed and compared

with the maximum value, as specified in the UNTIL option.

Phase 51 assigns identifying numbers to ON control cells using the ONCTR cell in COMMON. At execution time, each ON control cell will occupy four bytes in the ONCTL field of the Task Global Table, or TGT (see "Appendix B. Object Module" for the format of the TGT).

Registers 1 and 2 are generally used in the object program for nonarithmetic operations. When a nonarithmetic register is needed by a phase 51 verb, one will have been freed by phase 50's issuing of an instruction to store its contents in a subscript save cell, if necessary (see "Register and Storage Allocation" in the chapter "Phase 50" for a fuller description of register saving). Subscript save cell numbers are obtained from the SUBCTR cell in COMMON. They correspond to SUBADR cells in the Task Global Table of the object program. The instructions to save the registers would precede the LOAD instruction.

- ③ The literal 16 is not actually issued in the Procedure A-text. Rather, a literal definition element is issued, and the literal itself is written as Optimization A-text (see "Literals and Virtuals" in the chapter "Phase 50").
- ④ This branch statement transfers control to GN1 (the next source program statement) when ONCTR1 contains a value greater than 16.
- ⑤ This statement causes a branch to GN1 when ONCTR1 contains a value that is less than 2.
- ⑥ XSASW1 identifies a cell that will control the increment (EVERY option) of the ON statement at execution time. The switch will be flipped each time the statement is executed and tested to determine whether the imperative statement should be branched around. The switch is used only if the increment is 2 or in the ON 1 case.

<u>Source Statements</u>	
ON 2 AND EVERY 2 UNTIL 16 MOVE A TO B.	
ADD C TO D.	
<u>P2-text Strings</u>	
ON 2 2 16 GN1.	①
MOVE (2) A B.	
GN1. ADD (2) C D.	
<u>Procedure A-text</u>	
L 3,ONCTR1	②
LA 3,1(3)	
ST 3,ONCTR1	
C 3,=(16)	③
L 2,A(GN1)	
BCR NOTLO,2	④
C 3,=(2)	
BCR LO,2 5	
XI XSASW1,X'01'	⑥
CLI XSASW1,X'01'	
BCR NOTEQ,2	
instructions for MOVE	
GN1. instructions for ADD	

Figure 33. Analysis of an ON Statement

The identifying numbers are assigned from cell XSWCTR in COMMON. They correspond to cells in the XSASW field of the Task Global Table in the object program.

If the increment is not 2, an ON control cell is used to control the increment. Like the cell described in paragraph number 2, it is assigned an identifying number from ONCTR in COMMON, and it is used in a similar manner.

Special Considerations for Nonarithmetic Verbs

Some nonarithmetic verbs create special situations that require additional processing. These are described in this section.

Nonarithmetic Conversions: In a few instances, nonarithmetic data items must be expressed in binary during object program execution. These instances are illustrated by the following source program statements:

```
GO TO A B C DEPENDING ON X.
PERFORM RTNA X TIMES.
```

In both these cases, the value of X must be in binary when the statement is executed; however, the source programmer is not required to create X as a binary data item. This situation also arises in some Q-Routines.

To handle this, the verb analyzer calls routine DNTOR1. This routine determines whether the value is already in binary or must be converted. If conversion is needed, DNTOR1 generates code which converts the value at object time, and places the binary value in a work area in the TEMP STORAGE-2 field of the Task Global Table (leaving the value in the data area unchanged). The work area, rather than the data area, is then used when the value is referred to.

Procedure Branching in a Segmented Program:

If a GO statement transfers control out of the current segment and the current segment is not the root segment, phase 4 has indicated this by passing a special verb code (see "Checking for Segmentation in Procedure Branching" in the chapter "Phase 4"). When phase 51 encounters this verb code, it generates a call to the COBOL library subroutine, ILBOSGM0, or if the OPT option is in effect, to ILBOSGM1 (the calling sequence is given in the publication IBM OS/VS COBOL Subroutine Library Program Logic. This subroutine checks to see whether the necessary segment (the one containing the object of the GO TO) is already in storage, brings it into storage if it is not, initializes the segment, and transfers control to the named procedure.

If the operand is a PN or GN in a GO TO statement with the regular GO verb code, a normal branch is made whether or not the program is segmented. No test need be made, because phase 4 only issued a regular GO verb code if the branch did not require segment initialization.

If the operand is a VN, the SEGLMT cell in COMMON is tested. If the test indicates that the program is segmented (a value other than hexadecimal FF), a call to ILBOSGM0 is generated. If, at execution time, the VN is within the same segment, the subroutine will execute a normal branch. If the VN is not in the current segment, the subroutine will perform segment initialization.

If the GO TO statement contains a DEPENDING ON option, SEGLMT is tested to see whether the program is segmented. If it is, a call to the COBOL library subroutine, ILBOSGM0, is generated. If the OPT option is in effect, a call to the COBOL library subroutine, ILBOGD00, is generated. This subroutine passes control to the appropriate PN. If the program is segmented, the address of entry point ILBOSGM1 is passed in register 2 to the ILBOGD00 subroutine. This subroutine then determines which PN to branch to and passes control to entry point ILBOSGM1 to do standard processing for segmentation.

Verbs Requiring Calls to Object-Time Subroutines

In this section, the DISPLAY verb is discussed as an example of a verb which is executed by a COBOL library subroutine. (This discussion assumes that neither the DYNAM nor RESIDENT option is in effect and that the COBOL library DISPLAY subroutine, ILBODSS0, cannot be used.) The discussion is based on the DISPLAY statement shown in Figure 34. In this example, A, B, C, D, and E are data-names whose USAGE is DISPLAY and whose PICTURE is XX. The numbers of the following explanations refer to the circled numbers in Figure 34.

- ① Phase 4 puts a maximum of five operands in a string. Since the number of operands in this DISPLAY statement requires a continuation string, the first operand generated by phase 4 is the COBOL word FIRST, and the last operand is the COBOL word END. This is the form of all continued strings. The second operand identifies the device and the third through next-to-last operands are the data items named in the source statement. (Note that the P2-text contains the attributes, not the names, of the data items.)

```

Source Statement
  DISPLAY A B C D E UPON CONSOLE.

P2-text
  DISPLAY (5) FIRST CONSOLE A B C ①
  DISPLAY (3) D E LAST

Procedure A-text
  L 15,=V(ILBODSP0) ②
  BALR 1,15
  DC XL2'02' ③
  DC XL1'00' ④
  DC XL3,000002'
  DC AL4(BC-DISP)
  DC XL2'1F'
  .
  .
  DC X'FFFF' ⑤
    
```

Figure 34. Analysis of a DISPLAY Verb

- ② ILBODSP0 is the name of the COBOL library DISPLAY subroutine. Since it is a virtual (for a definition of virtuals, see the glossary), it is not written as part of the Procedure A-text. Rather, its virtual number is written as Procedure A-text; the name of the virtual itself is put out as Optimization A-text. See "DISPLAY Literals" below and "Literals and Virtuals" in the chapter "Phase 50"

for a discussion of how this text is produced.

- ③ This parameter gives the device code, which is 02 for CONSOLE. The section on ILBODSP0 in the publication IBM OS/VS COBOL Subroutine Library Program Logic, contains a complete list of device codes.
- ④ This parameter and the three which follow it give operand information for data-name A. Each operand is specified in a 10-byte field. The description of ILBODSP0 in the publication IBM OS/VS COBOL Subroutine Library Program Logic, gives all the codes; the meanings of the codes used in this example are as follows:

<u>Code</u>	<u>Meaning</u>
00	Specifies the type of the item. In this case, data-name A is nonnumeric, ready to display.
000002	Specifies the length of the item. Since the PICTURE for A is XX, the length is two bytes.
AL4(BC-DISP)	Specifies a displacement of an item from the beginning of the table identified by a base code.
1F	Specifies the displacement of A from the beginning of the area controlled by its base locator.

- ⑤ Following the description of A are similar 10-byte fields describing each of the other operands. The code FFFF follows the last description.

**Note:** As a special case, the DISPLAY of a single alphanumeric literal UPON CONSOLE generates an in-line WTO (write-to-operator) coding, rather than a call to the library subroutine.

If the STATE option has been specified, a call to the library subroutine ILBODBG4 is generated before any DISPLAY coding.

DISPLAY Literals

Generation of A-text, including literals and virtuals, is almost identical to that performed in phase 50. There is one exception, however. If a literal is in a

DISPLAY statement that requires a call to a COBOL library subroutine, a separate DISPLAY literal Optimization A-text element is written. This element is of a different type than an internal literal. It is generated differently so that phase 6 or 62 can build separate tables for internal and DISPLAY literals and search these tables using different techniques.

#### Generating System/370 Instructions

If a variable-length move or compare is required, the MVCL (for a move) or the CLCL (for a compare) machine instruction will be generated. If a compare involves a field greater than 256 bytes in length or if the receiving field for a move is greater than 512 bytes in length, the CLCL or MVCL machine instruction, respectively, is generated. If the receiving field for a move is right justified and the receiving field for the move is either greater than 512 bytes in length or variable in length, a call is generated to the ILBOSMVO COBOL library subroutine.

#### Text Generation for the OPT Option

If the OPT option is in effect, phase 51 generates the following text elements:

- Procedure A-text BC elements following a call to the COBOL library segmentation or GO TO DEPENDING ON subroutine.
- Procedure A-text C0 elements, specifically, C001, C003, and C006.
- Optimization A-text GNUREF (1C) elements, PNUREF (20) elements, GN-VN

(24) elements for PERFORM verbs, and VN (38) EQUATE (44) PN (4C) elements, and

#### GENERATING OBJECT CODE TO PROCESS VSAM FILES

Phase 51 contains a verb analyzer routine for each of the VSAM input/output verbs: OPEN, CLOSE, READ, WRITE, REWRITE, START, and DELETE. The VSAM verb routine:

- Analyzes the operands in the verb string.
- Creates the parameter list to be passed to the object-time subroutine that performs input/output operations for VSAM files.
- Generates the calling sequence for object-time subroutine.

The calling sequences to the COBOL object-time subroutines (ILBOVCO0, and ILBOVIO0) are described in IBM OS/VS COBOL Subroutine Library Program Logic.

#### GENERATING CALLS TO THE ILBOVCO0 AND ILBOVIO0 SUBROUTINES

The COBOL object-time subroutines act as the interface between the COBOL object program and VSAM. It uses the File Information Block (FIB) (built by phase 21), the File Control Block (FCB) (created at object-time), and the parameter list and options list (created in the calling sequence for the subroutine by phase 51). Phase 51 determines the parameters and the list of options by examining the verb string following each VSAM verb.

PHASE 6

Phase 6 (IKFCBL60) prepares a machine language program suitable for input to the linkage editor. The elements of this program are described in the chapter "Object Module." The phase is divided into several sequential parts, each of which performs specific functions. These are, in order:

- Determines object program storage allocation for the TGT (Task Global Table) by processing counters in COMMON and calculating the displacements of items that reside in the TGT at execution time.
- Optimizes literals, virtuals, source procedure-names, and compiler generated procedure-names by processing Optimization A-text and the PNTBL table; determining storage allocation in the PGT (Program Global Table) for these items and calculating their displacements, using counters in COMMON.
- With Procedure A-text as input, generates and writes machine language instructions. If the program is segmented, grouping the sections of instructions into segments. If the SXREF or the XREF option was specified, writes procedure-name and data-name definitions on DEF-text, and procedure-name and data-name references on REF-text, for input to phase 6A.
  - With Data A-text as input, writes object text for the data area of the object program.
  - Writes object text for the TGT and PGT from the RLDTBL table and Data A-text.
  - Writes object text for the INIT2, INIT3, and INIT1 routines of the object program, in that order.
  - Writes EXTRN statements for any program referenced by a CALL statement.
  - Passes E-text to phase 70 by storing it in the ERRBL table, or by writing it on SYSUT3 if the table overflows. If the TERM option was specified, incrementing the ERRNUM cell in COMMON each time a message definition element of E-text is encountered.

OUTPUT OF PHASE 6

The output of phase 6 depends on the compiler options specified by the user or determined by defaults set at installation time. The SXREF and XREF options have already been mentioned. The following are the other options that determine the output produced by phase 6:

<u>Option</u>	<u>Result</u>
PMAP	Causes the TGT, Literal Pool, PGT, register assignments, Working-Storage message, and a listing of the object text to be written on SYSPRINT.
CLIST	Causes the TGT, Literal Pool, PGT, register assignments, Working-Storage message, and a condensed object program listing to be written on SYSPRINT. The object program is limited to the card number, verb name (or verb number if the program is segmented), and address of the first instruction for each verb.
DMAP	Causes the TGT, Literal Pool, PGT, register assignments, and the Working-Storage message to be written on SYSPRINT. This option has already caused phase 3 to print a Data Division glossary.
LOAD	Causes the object program to be written on SYSLIN.
DECK	Causes the object program to be written (punched) on SYSPUNCH.
BATCH NAME	If both the BATCH and NAME options are specified, they cause a linkage editor control card to be generated at the end of the object program, so that the object program will be a separate load module. If BATCH is specified, the relative number of this compilation in the batch appears among the statistics printed by phase 6.

**Note:** The linkage editor control card generated for the BATCH and NAME options is produced by phase 65 if the FLOW or STATE option is specified.

**FLOW[=n[n]]** Causes the flow trace facility to be included in the object program. The number [n[n]] of traces requested is retained in the FLOWSZ cell in COMMON and is passed to phase 65, which places the number in the variable portion of the TGT.

**STATE** Causes the statement number facility to be included in the object program. Phase 6 writes Debug-text elements on SYSUT2 for use by phase 65.

For all compilations, compiler statistics are written on SYSPRINT from COMMON, where they were saved by phase 02.

The user may specify both the LOAD and DECK options, in which case the object program is written on both SYSLIN and SYSPUNCH. He may also specify NOLOAD and NODECK; in this case, he receives no executable copy of his object program.

If no output was requested (no PMAP, LOAD, DECK, CLIST, DMAP, BATCH, NAME, STATE, SXREF, XREF, VBREF, or VBSUM), phase 6 text processing is bypassed unless the TERM option was specified. In this case, phase 6 scans the E-text on SYSUT4 and increments the ERRNUM cell in COMMON. After phase 6 scans the E-text, it rewinds SYSUT4 and returns to phase 00. Phase 00 uses the count in ERRNUM to write a message to SYSTEMM giving the number of errors encountered for the compilation. Phase 6 also sets a bit in COMMON to indicate whether phase 70 is required (see "Suppression of Output Listing" below).

#### Suppression of Output Listing

If the SUPMAP (suppress map) option is in effect, no output is produced by phase 6 if a D-level or E-level error message was generated by any phase. This is determined

by testing the ERRSEV cell in COMMON. A value of 12 or greater means that at least one D-level or E-level message occurred.

The ERRSEV cell was set by phases 2, 3, 4, 50, and 51 every time they encountered or generated an element of E-text (see "E-text" in the chapter "Phase 51"). A test is made for this condition upon entering phase 6. If it occurs, a message is printed and the text is not processed unless SXREF or XREF is specified.

If the SUPMAP condition occurs and neither SXREF or XREF was requested, phase 6 terminates processing after it sets two bits in COMMON (bits 6 and 7 of the second byte of SWITCH). Bit 6 indicates that phase 70 is to be called, and bit 7 indicates to phase 70 that E-text must be read from SYSUT4. However, if the TERM option was specified, phase 6 scans the E-text on SYSUT4 and increments the ERRNUM cell in COMMON. After phase 6 scans the E-text, it rewinds SYSUT4 and returns to phase 00.

If SXREF or XREF was requested, text is processed but the only output is REF-text and DEF-text (which phase 6A uses to produce the cross-reference listing). The ERRTEL is built, or E-text is written on SYSUT3. A bit is set in COMMON (bit 6 of the second byte of SWITCH), indicating to phase 6A that phase 70 is required. If E-text was written on SYSUT3, bit 5 of the second byte of SWITCH is set to indicate that E-text must be read from SYSUT3.

Phase 6 does not write object text in execution-time sequence. Rather, it instructs the linkage editor to reorder the text by assigning relative addresses. To do this, it allocates space for areas that will be written later, incrementing the LOCCTR (location counter) cell of COMMON to reflect the relative location at execution time of the area currently being processed.

#### Glossary and Listing Symbols

The symbols used in the Listing and Glossary to define compiler-generated information are shown in Figure 35.

Symbol	Definition	Description
BL	Base Locator	See "BL" in "Task Global Table"
BLL	Base Locator for Linkage Section	See "BLL" in "Task Global Table"
CKP	Checkpoint Counter	See "CHECKPT CTR" in "Task Global Table"
DBGC	Debug Card Number	See "DEBUG CARD" in "Task Global Table"
DBGI	Debug Information Pointer	See "DEBUG PTR" in "Task Global Table"
DBGT	Debug Transfer	See "DEBUG TRANSFER" in "Task Global Table"
DCB	DCB Address	See "DCBADR" in "Program Global Table"
DEC	DECB Address	See "DECBADR" in "Task Global Table"
DNM	Source Data Name	See "Glossary Building"
FIB	FIB Address	See "FIB" in "Task Global Table"
GN	Generated Procedure-name	See "GN" in "Program Global Table"
INX	Index Cell	See "IND" in "Task Global Table"
LIT	Literal	See "Literal" and "Display Literal" in "Program Global Table"
ON	ON Counter	See "ONCTL" in "Task Global Table"
OVF	Overflow Cell	See "OVERFLOW" in "Program Global Table"
PBL	Procedure Block (Optimizer)	See "PROCEDURE BLOCK" in "Program Global Table"
PFM	PERFORM Counter	See "PFMCTL" in "Task Global Table"
PN	Source Procedure-name	See "PN" in "Program Global Table"
POV	PGT Overflow	See "OVERFLOW" in "Program Global Table"
PRM	Parameter	See "PARAM" in "Task Global"
PSV	PERFORM Save	See "PFMSAV" in "Task Global Table"
RSV	Report Save Area	See "RPTSAV" in "Task Global Table"
SAV	Save Area Cell	See "Save Area" in "Task Global Table"
SA2	Input/Output Error Save Cell	See "Save-Area -2" in "Task Global Table"
SA3	OPEN Parameter	See "Save-Area-3" in "Task Global Table"
SBL	Secondary Base Locator	See "SBL" in "Task Global Table"
SBS	Subscript Address	See "SUBADR" in "Task Global Table"
SSVE	Sort Save Area	See "Sort Save" in "Task Global Table"
SWT	Switch Cell	See "Switch" in "Task Global Table"
TLY	Tally Cell	See "Tally" in "Task Global Table"
TOV	TGT Overflow	See "OVERFLOW" in "Task Global Table"
TS	Temporary Storage Cell	See "TS" in "Task Global Table"
TS2	Temporary Storage (Non-Arithmetic)	See "TS-2" in "Task Global Table"
TS3	Temporary Storage (Synchronization)	See "TS-3" in "Task Global Table"
TS4	Temporary Storage (Table-Handling)	See "TS-4" in "Task Global Table"
V (BCDNAME)	Vitrual	See "Virtual" in "Program Global Table"
VIR	Virtual Cell	See "Virtual" in "Program Global Table"
VLC	Variable Length Cell	See "VLC" in "Task Global Table"
VN	Variable Procedure-name	See "VN" in "Task Global Table"
VNI	Variable-name Initialization	See "VNI" in "Program Global Table"
WC	Working Cell	See "Working Cells" in "Task Global Table"
XSA	Exhibit Save Area	See "XSA" in "Task Global Table"
XSW	Exhibit Switch	See "XSASW" in "Task Global Table"

Figure 35. Symbols Used in the Listing and Glossary to Define Compiler-Generated Information

TASK GLOBAL TABLE STORAGE ALLOCATION

When phase 6 receives control, the LOCCTR cell contains the relative address of the Task Global Table (TGT) in the load module. LOCCTR was set by phases 22 and 21, which added the length of the data area to that

of the INIT1 routine. (These areas precede the TGT in the load module.)

Routine TGTINT first does preliminary computations to determine the length of the entire TGT. If this length exceeds 4096 bytes, one 4-byte OVERFLOW cell is allocated for each 4096-byte area after the

first. Then this routine computes the locations of TGT fields after the OVERFLOW cells.

Some fields of the TGT are constant in length; others are variable, depending on the requirements of the program being compiled. For most of the variable fields, there is a counter in COMMON used to compute its length. When the value has been used, the counter is set to the displacement of the current field in the TGT. Figure 36 lists these counters and the TGT fields to which they correspond. In a register called RW1, a counter is kept of the displacement of the current field in the TGT.

Some of the counters in COMMON specify a number of bytes. Others specify a number of entries, where each entry requires two or eight bytes. In the latter case, the value of the counter is multiplied by 2 or 4 before it is used to compute displacements.

This is done in routine DSPLAC, which is called for each variable-length field. Two parameters are passed to this routine: the address of the counter in COMMON, and the number of bytes for each entry. From the number of bytes, DSPLAC also determines boundary alignments. DSPLAC places the value of RW1 (the displacement of the field in the TGT) into the counter, and adds the length of the field to RW1.

If the PMAP, CLIST, or DMAP options are in effect, DSPLAC calls routine MAPLOC, which prints one line at a time.

If the statement number option (STATE) or the flow trace option (FLOW) was specified, the SWITCH cell, the current priority cell (for STATE only), the TGTTAB pointer, and the TGTTAB information are set by phase 65.

After the length of the entire TGT has been calculated, the value of RW1 (the length of the TGT) is added to the LOCCTR cell. The value of the LOCCTR cell is now the displacement of the PGT.

#### OPTIMIZING STORAGE FOR THE PROGRAM GLOBAL TABLE

The general function of this part of phase 6 is to allocate space for the Program

Global Table (PGT) in the same way that TGT storage was allocated. Before this can be done, however, the required space must be determined for the literals, virtuals, and procedure-names that reside in this table at execution time. The routines that determine the lengths of these fields also optimize the contents of the fields by eliminating duplication.

For optimizing, the PNTBL table and Optimization A-text are used. The processing of the PNTBL table occurs first. Then the Optimization A-text is read and processed.

Optimization A-text, which was generated by phases 50 and 51, contains the following kinds of elements:

- EQUATE strings, which equate any procedure-names (PNs) and generated procedure-names (GNs) that refer to the same location (for a description of how and why these strings are built, see "Building PN and GN Equate Strings" in the chapter "Phase 51").
- Literal definitions, containing the actual value of the literal.
- DISPLAY literal definitions.
- Virtual reference definitions for input/output error routines.
- Virtual definitions.
- Variable procedure-name (VN) definitions, if the program is segmented.

#### Building the VN Priority Table

VN definition elements are not used for optimization. They are included in the Optimization A-text for a segmented program because they are used to build a table (called VNPTY) which must be in storage for the next part of phase 6 processing. As an element is read, it is entered unchanged into this table. After the Optimization A-text file has been closed, the VNPTY table is sorted in ascending order of priority number.



Counter	Contents From Earlier Phases	TGT Field	Multiplication Factor
TSMAX	Number of bytes needed for arithmetic temporary storage.	TEMP STORAGE	8
TS2MAX	Number of bytes needed for nonarithmetic temporary storage.	TEMP STORAGE-2	1
TS3MAX	Number of bytes of work area for aligning non-SYNCHRONIZED data items.	TEMP STORAGE-3	1
TS4MAX	Number of bytes of work area for table-handling verbs.	TEMP STORAGE-4	1
BLLCTR	Number of base locators assigned to Linkage Section.	BLL	4
VLCCTR	Number of variable-length cells (containing current length of a variable-length field).	VLC	2
INDEX1	Number of index-names defined in INDEXED BY clause.	INDEX	4
SBLCTR	Number of secondary base locators (location of a field variably located because it follows a variable-length field).	SBL	4
BLCTR	Number of base locators assigned to files and Working-Storage.	BL	4
SUBCTR	Number of subscript save cells.	SUBADR	4
ONCTR	Number of ON control cells.	ONCTL	4
PFMCTR	Number of PERFORM control cells (for PERFORM X TIMES).	PFMCTL	4
PSVCTR	Number of PERFORM save cells.	PFMSAV	4
VNLOC*	Number of variable procedure-names.	VN	8
DECBCT	Number of DECBs.	DECBADR	4
XSNCTR	Number of EXHIBIT switches.	XSASW	1
XSACTR	Number of bytes for EXHIBIT saved area.	XSA	1
PARMAX	Area needed for parameter lists.	PARAM	4
AMICTR	Number of FIBs for VSAM files.	FIB	4
<p>*The number of VNs in the program is passed to phase 6 in the VNCTR cell of COMMON, not the VNLOC cell. However, this value is moved into the VNLOC cell and all further TGT processing uses VNLOC rather than VNCTR. (The number of VNs in the program must also be known for PGT allocation to determine the size of the VN field in the PGT. This value is saved in VNCTR.)</p>			

Figure 36. Use of Counters in COMMON to Allocate Space in the TGT for Variable-length Fields (Part 1 of 2)

Counter	Contents From Earlier Phases	TGT Field	Multiplication Factor
RPTSAV	Report Writer save area requirements.	REPORT SAVE	4
CKPCTR	Number of checkpoint requests.	CHECKPT CTR	4
SA2CTR	USE LABEL or USE ERROR procedure save area requirements.	SAVE AREA-2	4
SA3CTR	Maximum number of files specified in an OPEN statement.	SAVE AREA-3	4

Figure 36. Use of Counters in COMMON to Allocate Space in the TGT for Variable-length Fields (Part 2 of 2)

Optimizing PNs and GNs

The first step in optimizing PNs and GNs is to allocate space in the compiler table area for the PNTBL and GNTBL tables. The lengths of these tables are determined from the values of PNCTR and GNCTR in COMMON, respectively. Then, in routine PNUPRO, the PNTBL is processed against the PNTBL. (The PNTBL, containing one entry for each procedure-name in the program, is used only in phase 6; the PNTBL was built by phase 51. See "Building the PNTBL Table" in the chapter "Phase 51", for a description of how and why this table was created.) If a PNTBL entry has a value of 1, the corresponding PNTBL entry is numbered. The numbers are sequential beginning with 1. If the PNTBL entry has a value of 0, this means that the procedure-name is never referred to in the program and can be eliminated; therefore, the corresponding PNTBL entry is set to 0. Once the PNTBL values have all been set, the PNTBL table is released.

Figures 37 through 39 show an example of this processing for a program containing six PNs and six GNs. In Figure 37, the table entries are shown as they would appear after the PNTBL processing.

Optimization A-text is then read by routine READF2. Each time a PN or GN Equate string is encountered, READF2 calls the routines (PNEQR or GNEQR, respectively) that process these strings.

	PNUTBL	PNTBL	GNTBL
1	1	1	0
2	0	0	0
3	1	2	0
4	1	3	0
5	0	0	0
6	1	4	0

Figure 37. PNTBL, PNTBL, and GNTBL Tables at the Beginning of Optimization Processing

Figure 38 shows the effect of a PN EQUATE string indicating that PN3 (the number found in the PNTBL entry for PN3, 2 in the example) is entered into the slot for GN1. If there were no other Equate strings read, the following would occur after the Optimization A-text file had been closed: GN2 through GN6 would be assigned relative numbers sequentially, starting with the number after the last referenced PN number in the PNTBL table (which is 4 in the example). The GNTBL entries would then read 2, 5, 6, 7, 8, 9.

If, however, as shown in Figure 38, a GN equate string is encountered equating GN2 with GN4 and GN5, the relative number of GN2 is assigned to GN4 and GN5. This number will be 5, since GN2 will contain the next sequential number after PN6.

GNTBL		
1	2	Equates GN1 to PN3
2	5	
3	0	
4	5	Equates GN2, GN4, and GN5, which it can be assumed will be assigned a relative number of 5.
5	5	
6	0	

Figure 38. GNTBL Table after PN and GN Equate Strings Have Been Processed

After the Optimization A-text file has been closed, relative numbers are assigned to each GN not equated to a PN or another GN. The completed GNTBL table for the example is shown in Figure 39.

GNTBL	
1	2
2	5
3	6
4	5
5	5
6	7

Figure 39. GNTBL Table after the Relative Numbers Have Been Assigned

**Note:** In the object code listing, the optimized GNs are numbered sequentially starting with 1. The numbers to the left of the tables in Figures 37, 38, and 39 are A-text PN and GN numbers before optimization. They represent implicit positions in the tables.

Optimizing Literals and DISPLAY Literals

The literal optimization routines are used to eliminate storage duplication in cases where the source programmer used the same literal more than once. Routine LTLRTN processes internal literals, and routine LTLDIS processes DISPLAY literals. These

routines build three tables: the CONTBL and CONDIS tables (for regular and DISPLAY literals, respectively) contain one entry for each unique literal, and the LTLTBL table contains an entry for each use of a literal.

When a literal definition is encountered, the CONTBL or CONDIS table is searched for an entry identical to the literal. (To be identical, two internal literals must meet the same boundary requirements as well as have the same value.) If no match is found, the new literal is entered into the CONTBL or CONDIS table. Any bytes skipped because of boundary alignment are filled with zeros. The displacement of this entry from the beginning of the table is placed in the LTLTBL table, with a bit set to indicate whether it is a CONTBL or CONDIS entry. If a match is found, only an LTLTBL entry is made. This LTLTBL entry is the displacement of the CONTBL or CONDIS entry that matched the literal being processed.

Figure 40 shows an example of these tables after all Optimization A-text has been processed. The Optimization A-text contained literal definition elements for the following literals:

8, 3(DISPLAY), 3, 9, Y(DISPLAY), 8

After the Optimization A-text data set is closed, the Literal Pool is written on SYSPRINT, using the contents of the CONTBL and CONDIS tables, if the PMAP, CLIST, or DMAP options are in effect. The Literal Pool is also written on SYSPUNCH and SYSLIN, as needed.

Optimizing Virtuals

The virtual optimization routine (VIRRTN) is used to eliminate storage duplication in cases where the same EBCDIC name of a called program is referred to in more than one CALL statement or in more than one call to a COBOL library object-time subroutine. The logic of this processing is similar to that of literal optimization. Two tables are built: the CVIRTB table contains one entry for each unique virtual, and the VIRPTR table contains one entry for each reference to a virtual.

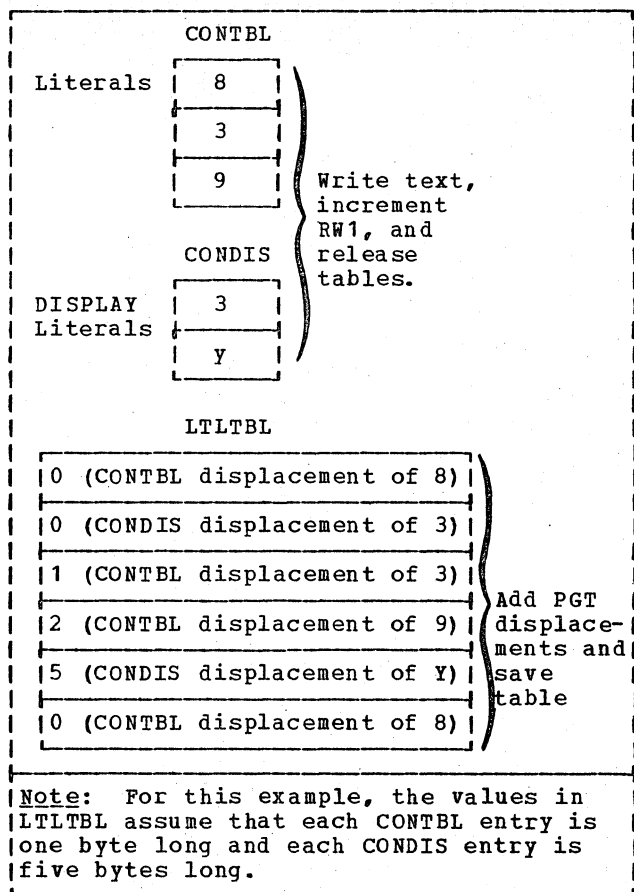


Figure 40. CONTBL, CONDIS, and LTLTBL Tables after Processing Literals

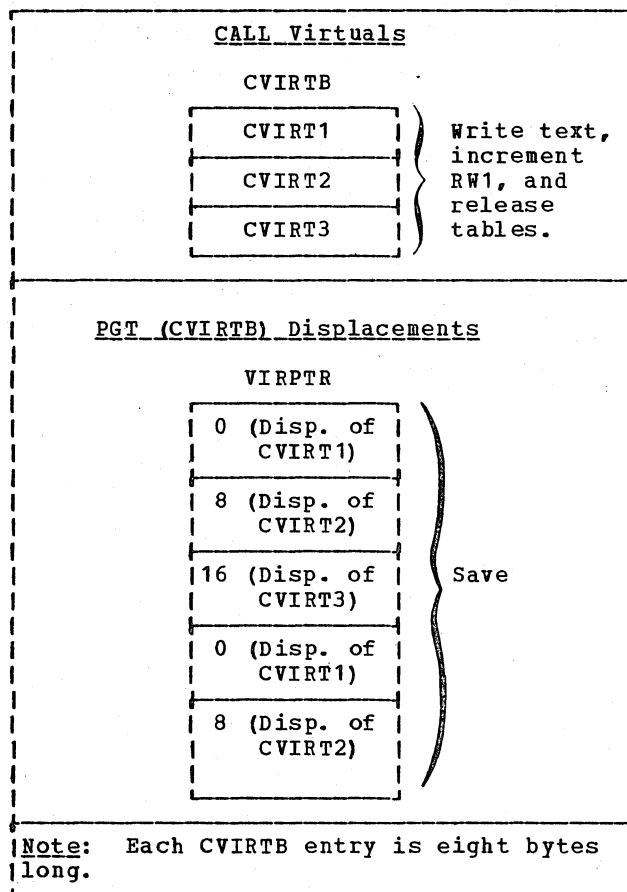


Figure 41. CVIRTB and VIRPTR Tables after Processing Virtuals

When a virtual definition is encountered, the CVIRTB table is searched for an identical entry. If none is found, the new virtual is entered in the CVIRTB table. Into the VIRPTR table is entered the displacement of this virtual from the beginning of the CVIRTB table. If a match is found, only a VIRPTR entry is made. This VIRPTR entry contains the displacement in the CVIRTB table of the entry that matched the virtual being processed.

Figure 41 shows the contents of these tables after processing Optimization A-text for a program containing the following virtuals:

CVIRT1, CVIRT2, CVIRT3, CVIRT1, CVIRT2.

ALLOCATING STORAGE FOR THE PROGRAM GLOBAL TABLE

When all Optimization A-text has been read, storage is allocated for the PGT and PGT initialization is done. Entries for the External Symbol Dictionary are created for virtuals, and object text is written for virtuals and literals. Register RW1 is used throughout PGT allocation to hold the displacement of the field currently being processed. Counters in COMMON are set to the displacements of their corresponding PGT fields from the beginning of the PGT.

If the PMAP, CLIST, or DMAP options are in effect, the format of the PGT is written on SYSPRINT using routine MAPLOC.

OVERFLOW Allocation

Preliminary calculations are made to determine whether the size of the PGT exceeds 4096 bytes. If it does, one 4-byte OVERFLOW cell is required for each 4096-byte area after the first one.

VIRTUAL Allocation

After the OVERFLOW CELLS field of the PGT has been calculated, the VIRTUAL field is processed. Using the CVIRTB table, an External Symbol Dictionary entry (ESD-text type 2) is written for each virtual unless the DYNAM or the RESIDENT option is in effect. (If the RESIDENT option is in effect, no ESD or RLD item is written for a library subroutine; if the DYNAM option is in effect, no ESD or RLD item is written for a library subroutine or a user subprogram.) Object text is also written and entries are made in the RLDTBL table for subsequent writing of the Relocation Dictionary.

The VIRCTR cell of COMMON is set to the displacement of the VIRTUAL field from the beginning of the PGT. (This value is 0 unless OVERFLOW cells have been allocated.)

To determine the length of the VIRTUAL field, four bytes are allowed for each entry in the CVIRTB table. The calculated length is added to register RW1. If the DYNAM and/or RESIDENT option is in effect, the CVIRTB table is used to enter in the PGT the EBCDIC names of those routines that are to be dynamically loaded. The CVIRTB table is kept for use during Procedure A-text processing, when its contents (EBCDIC names) will be used to generate comments for CALL statements.

The entries in the VIRPTR table are changed to contain displacements in the VIRTUAL field (see the example in Figure 42). The table is saved for subsequent use during Procedure A-text processing.

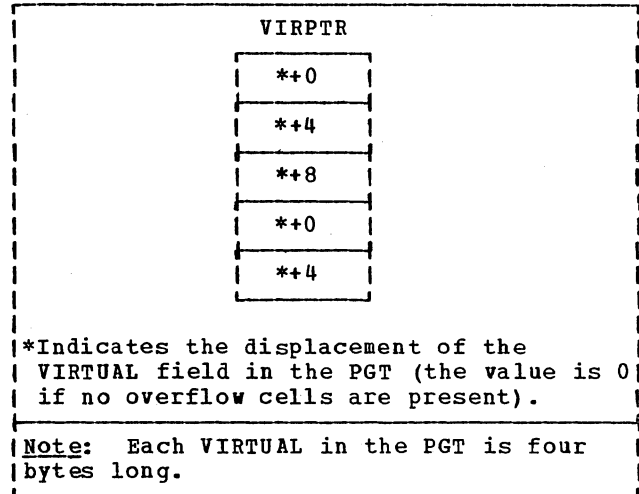


Figure 42. VIRPTR Table after VIRTUAL Allocation

VIRTUAL EBCDIC NAMES Allocation

If the DYNAM or RESIDENT option is in effect, an 8-byte cell for each library subroutine name is allocated in the PGT. In addition for the CALL identifier, or if the DYNAM option is in effect for the CALL literal, an 8-byte cell for each user subprogram name is allocated in the PGT. If neither option is in effect, this field does not exist.

A count of the EBCDIC names to be placed in the PGT is kept in the BCDCTR cell in COMMON. This count is multiplied by 8 to reserve space for the list of names. After VIRTUAL allocation, register RW1 contained the displacement of the VIRTUAL EBCDIC NAMES field; the displacement is saved in the BCDISP cell in COMMON and register RW1 is incremented to reflect the allocated bytes for the VIRTUAL EBCDIC NAMES cells.

PN Allocation

After the VIRTUAL field or the VIRTUAL EBCDIC NAMES field, if allocated, has been processed, the value in register RW1 is the displacement of the PN field in the PGT. This value is saved in the PNCTR cell of COMMON.

For each referenced PN in the PNTBL table, a 4-byte cell is allocated in the PGT. The PNTBL entry is set to the displacement of this cell from the beginning of the PGT. If a PN was not referenced (if the value in the PNTBL entry

was 0), no space is allocated. In the example in Figure 37, only four 4-byte cells are required in the PGT. After the PNTBL table entries have been adjusted, the entry for PN3 exceeds the entry for PN1 by four, and the entry for PN2 remains 0. The total length of the PN field (16 bytes in the example) is then added to RW1.

Figure 43 shows the PNTBL table for the same program as in Figure 37, after PN allocation in the PGT. The table is saved for use during Procedure A-text processing, when the values it contains will be used as displacements in instructions.

PNTBL		GNTBL	
1	** *+0	1	** *+4
2	0	2	** *+16
3	** *+4	3	** *+20
4	** *+8	4	** *+16
5	0	5	** *+16
6	** *+12	6	** *+24

\*Indicates the displacement in bytes of the PN field from the beginning of the PGT.

**Note:** The numbers to the left of the tables are A-text PN and GN numbers. They specify implicit positions in the table.

Figure 43. PNTBL and GNTBL Values after PGT Allocation

GN Allocation

The value in register RW1 is now the displacement of the GN field in the PGT. This value is placed in the GNCTR cell of COMMON.

For each unique GN, a 4-byte cell is allocated in the PGT. The GNTBL entry is set to the displacement of this cell from the beginning of the PGT. However, if the GN was equated to a PN or another GN, the GNTBL entry is set to the PGT displacement of that PN or GN. This is illustrated in Figure 43, which shows the GNTBL table after this processing using the same example as in Figures 37, 38, and 39. In this example, GN4 and GN5 were equated to GN2. Therefore, the GNTBL entries for GN4 and GN5 contain the displacement of GN2. The PGT for this program will contain only

three unique GN entries, or twelve bytes. After all entries have been processed, the length of the GN field (12 in the example) is added to register RW1.

The GNTBL table is saved for use during Procedure A-text processing.

DCBADR Allocation

The DCBCTR cell in COMMON is used to determine how much space is required: four bytes are reserved for each DCB in the program. The DCBCTR cell is set to the displacement of the DCBADR field (the value of register RW1), and RW1 is then incremented to reflect the allocated bytes.

VNI Allocation

The VNCTR cell of COMMON was used by earlier phases to count the number of variable procedure-names in the program. Eight bytes are allocated for every VN. The displacement of the VNI field (the value of RW1) is placed in the VNILOC cell, and the number of bytes allocated is added to register RW1.

LITERAL Allocation

The displacement of the LITERAL field in the PGT (the current value of register RW1) is placed in the LTLCTR cell of COMMON. The length and contents of the LITERAL field will be identical to the CONTBL and CONDIS tables.

For each LTLTBL entry that refers to CONTBL, the value in the LTLTBL table is replaced by the displacement of the specific literal from the beginning of the PGT. This displacement is calculated by adding the value already in the LTLTBL entry (which is the CONTBL displacement of the literal) to the value of RW1. An example is shown in Figure 44.

The same processing occurs for LTLTBL entries that refer to CONDIS, except that the increment includes the length of the CONTBL table. This occurs because DISPLAY literals are placed after internal literals in the PGT, as illustrated by Figure 44. The LTLTBL table is saved for use during Procedure A-text processing. The lengths of the CONTBL and CONDIS tables are used to increment RW1. If the program is not segmented, the contents of the tables are

used to write object text, and the tables are released. In a segmented program, the writing of object text is delayed and, therefore, the tables are kept.

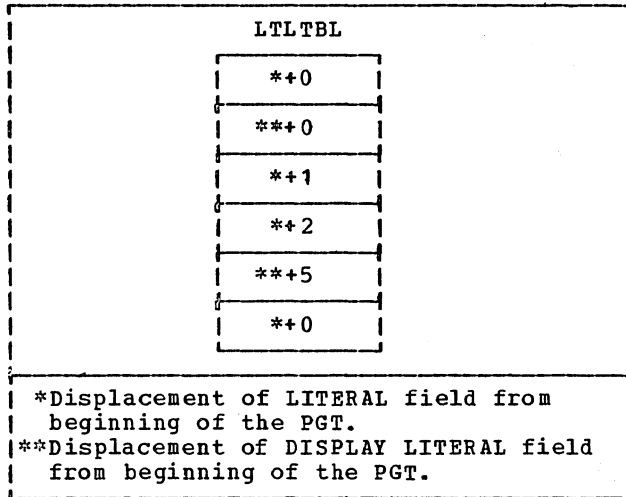


Figure 44. LTLTBL Table after Literal Allocation

PROCEDURE A-TEXT PROCESSING

Phase 6 reads Procedure A-text to produce machine-language instructions for the object program. One element of text is read and processed at a time, and the object code produced for this element is placed in a work area called OU6REC. One or more elements are required to produce a complete instruction. When an instruction is complete, it is written out from the work area, and the LOCCTR cell in COMMON is incremented by the number of bytes written.

If the instruction involves a base locator, the processing routine refers to or updates table REGMTX (see "Execution-Time Base Register Assignment" in this chapter), which is a table internal to phase 6 (that is, not a TAMER table). Base locators were assigned by phase 22; a discussion of their meaning appears in the chapter "Phase 22."

If the PMAP compiler option was specified, routine PUT is called to write a line of text on SYSPRINT every time a complete instruction has been created. If the CLIST option was specified, this routine is called only for each source program verb.

If the STATE option is in effect, Procedure A-text is used to create Debug-text which is written on SYSUT2. Debug-text elements are written by the SYS2 routine for all card numbers encountered and contain the card number and its displacement within the object module. This text is used by phase 65 to produce the PROCTAB and SEGINDX tables which are written in the object module. If either the FLOW or the STATE option is in effect, phase 6 builds the TGTADTBL table which is used to pass debugging information to phase 65.

If the SXREF, XREF, VBREF, or VBSUM option is in effect, Procedure A-text is used to create REF-text and to write it on SYSUT3. This text, containing an element for every data-name, file-name, procedure-name, and verb in the program, is used by phase 6A to produce a cross-reference listing.

Figure 45 describes the processing for each type of Procedure A-text element. The individual elements are illustrated in "Section 5. Data Areas."

Note on base registers for the PGT and TGT: At execution time, R12 always points to the beginning of the PGT and R13 always points to the beginning of the TGT. If the displacement of an item in the PGT or TGT exceeds 4096 bytes, an OVERFLOW cell must be used. The OVERFLOW cells fields of both the PGT and TGT are at fixed displacements from R12 and R13, respectively. Which OVERFLOW cell is to be used is determined from the value of the displacement, for example, a value from 4096 to 8191 bytes uses cell 1, from 8192 to 12,287 bytes uses cell 2, etc. An instruction is generated to load R14 or R15 from the OVERFLOW cell. Then, in the operand currently being processed, R14 or R15 is used as the base, and the displacement is decremented by 4096, 8192, etc.

Code and Type	Action Taken
2C* card number	Store in 3-byte cells OU6C0N and XPCDNO. If PMAP or CLIST are requested, read Listing A-text. Used to create an in-line constant for TRACE instructions which call the DISPLAY object-time subroutine (ILBODSP0). If STATE is requested, write Debug-text.
30* PN definition	Using PN number as an index, look in PNTBL (see "PN Allocation" in this chapter) to get displacement in PGT of the cell for this PN. Create an RLDTBL entry which will place the current value of LOCCTR in the PGT cell.
34* GN definition	Same as PN definition, using GN number and GNTBL (see "GN Allocation" in this chapter).
38* VN definition	Create an indirect RLDTBL entry from this element and the PN reference which follows it.
3C EBCDIC card name	Convert the current card number to an EBCDIC constant of the form: DC X'5' DC CL6'generated card number'
44 macro-type instruction	Use byte 2 of the element as index to a branch table. Phase 6 produces the required coding. The contents of these elements are listed in the Procedure A-text formats of "Section 5. Data Areas."
*Indicates that no object text was written for this element.	
**See note under "Procedure A-text Processing."	

Figure 45. Processing Procedure A-text Elements (Part 1 of 3)



Code and Type	Action Taken
48 operation code	This element contains, in machine language the first two bytes of an instruction. The first byte is the operation code; the second may give condition codes, registers, or other operands. For an RR type instruction, this element contains the complete instruction. It is written out as received.
4C PN reference	This is the operand of a LOAD instruction. Procedure branching is accomplished by loading an address and then branching to it. Using register 12** as a base, find displacement by using PN number as an index into PNTBL (see "PN Allocation" in this chapter). Using card number stored in XPCDNO, write an element of REF-text for phase 6A, if SXREF or XREF is in effect.
50 GN reference	Same as PN reference, using GN number and GNTBL (see "GN Allocation" in this chapter). No REF-text is written.
54 VN reference	Use register 13** as base. Get displacement of VN field of TGT from VNLOC cell in COMMON (see "Task Global Table Storage Allocation" in this chapter). Use a VN number to compute displacement of this VN cell.
58 Virtual reference	Use virtual number as an index in the VIRPTR table (see "VIRTUAL Allocation" in this chapter). Table entry contains displacement of this virtual in the PGT. Use register 12** as a base.
5C BL reference	This element is the operand of an instruction which loads a base register. Use register 13** as a base. Get displacement of BLL or BL field in TGT from BLLCTR or BLCTR, respectively, in COMMON. Use BL number to compute displacement of this cell. Update table REGMTX.
60 TGT standard area reference	Use register 13** as a base. Displacement is picked up from a list of constants. This element refers to a cell in the fixed portion of the TGT.
64 Global Table variably- located area reference	Use register 13** as a base (unless the element specifies the DCBADR field of the PGT, which uses register 12**). Get displacement of the TGT or PGT field from the appropriate cell in COMMON, and use identifying number to compute displacement of this item (see "Task Global Table Storage Allocation" and Figure 36 in this chapter).
68 literal reference	Bytes 2 and 3 are used to find the correct entry in the LTLTBL table, which gives the displacement of this literal in the PGT. Register 12** is the base.
6C DC definition	This element is used to create an inline constant for a calling sequence. It is always preceded by the element 4424, the macro-type instruction element signaling a DC definition (see Code 44 above in this figure). It is written out as received.
70 base and displacement	Specifies the actual register number and displacement for the instruction. It is written out as received.
<p>*Indicates that no object text was written for this element.  **See note under "Procedure A-text Processing."</p>	

Figure 45. Processing Procedure A-text Elements (Part 2 of 3)

Code and Type	Action Taken
78 address reference	Search table REGMTX on <i>i</i> and <i>k</i> (BL type and BL number). If a match is found, the required BL is already in a register. Use that register as the base. If a match is not found, generate an instruction to load register 14 or register 15 with the BL from the BL or BLL field of the TGT and use that register as the base (see Code 64 above for generation of the LOAD macro instruction). Displacement is the <i>d</i> field of the element. Get card number from XFCNO to write an element of REF-text.
7C EBCDIC data-name reference	This element always follows the element 4404, the macro-type instruction element for ENTRY. It is used to punch an ESD-text type 1 card for the entry point.
80 address increment	This element is required, for example, by the second MVC for a MOVE of more than 256 bytes. The element itself would have a value, in this case, of 256 bytes (the value of the increment). Add it to the <i>d</i> (displacement) field of the preceding reference.
84 relative address	This element is used to create an inline pointer to an item in a field of the TGT or PGT for a calling sequence. Get displacement of field from appropriate counter in COMMON and use identifying number to compute displacement of item.
A0* register specification	Specifies the register used by a macro-type instruction element, and must follow certain of these elements (see the list of macro-type instructions under "Procedure A-text" in "Section 5. Data Areas").
A4 incremented address	This element combines the address reference and increment elements into one (see Codes 78 and 80 above).
B0 calling sequence displacement	Used to create an in-line TGT or PGT pointer for a call to an object-time subroutine which requires a parameter containing a displacement from register 13 or register 12.
B4* calling sequence dictionary pointer	Used when a file-name or data-name occurs in a calling sequence to write a REF-text element for phase 6A. Obtain card number from XFCNO.
B8* file reference	Used to write an element of REF-text.
*Indicates that no object text was written for this element. **See note under "Procedure A-text Processing."	

Figure 45. Processing Procedure A-text Elements (Part 3 of 3)

Procedure A-text Processing in a Segmented Program

Phase 6 reads Procedure A-text in the order in which it was written.

Procedure A-text is read from the direct-access data set SYSUT1 using the segment priority (SEGTBL) table. For a description of how this table is built, see "Segmentation Control Breaks" in the chapter "Phase 51." The table format is given in "Section 5. Data Areas."

In phase 1B, the priority numbers of all sections in the root segment were set to 0.

Routine SEGPROC searches the SEGTBL table for the first entry whose priority is zero. It then calls COS in phase 00 with a request for SEGPNT, passing the relative disk address of this section. The SEGPNT routine in phase 00 positions the access mechanism to the correct address on the file. (Additional information on this routine is in the chapter "Phase 00" under "Phase Input/Output Requests.") The section of Procedure A-text is then read and processed. When a segmentation control break is encountered in the text, the SEGTBL is searched for other sections of the same priority.

Note: A section is a series of source program procedure instructions grouped under the same section-name. A segment is all the instructions whose sections have the same priority, and a segment may consist of one or more sections. There is a SEGTBL entry for every section whose priority differs from that of the section preceding it.

When all sections of one priority have been processed, the SEGTBL table is searched for a different priority, and the process is repeated. If the STATE option was specified, at the end of processing for each segment, the final LOCCTR value for that segment and the priority for the next segment to be processed are both written on SYSUT2 for phase 65. Object text for the segments is written throughout this processing, as machine instructions are generated. If any segment refers to another program via a CALL statement, an INSERT card for the called program is generated also.

After the last nonroot segment has been processed, the LOCCTR cell of COMMON is set to the location in the root segment of the PGT. Object text is then written from the CVIRTB, CONTBL, and CONDIS tables, which contain the values of virtuals and literals to be stored in the PGT (see "LITERAL Allocation" and "VIRTUAL Allocation" in this chapter). Then LOCCTR is set to the beginning of the Procedure Area of the root segment, which was saved in cell LOCPGM, and processing of the Procedure A-text for the root segment begins. The text is located on SYSUT1 by finding all entries of zero priority in the SEGTBL table.

Listing A-Text

Listing A-text is processed concurrently with Procedure A-text. Generated by phase 1B, it contains the EBCDIC names of procedure-names and COBOL verbs preceded by a code and a count. It is used to print procedure and verb-names alongside their associated code on the object program listing (if the PMAP option is in effect), to print verb-names (if the CLIST option is in effect), and in response to READY TRACE (if the VERB option is in effect). One Listing A-text element is read every time a card number element is encountered in Procedure A-text.

### Execution-Time Base Register Assignment

Before Procedure A-text processing begins, permanent base registers are assigned. Register 12 is always assigned to the PGT, and register 13 to the TGT. Registers 6 through 11 are available to the data area. Of these registers, register 6 is permanently assigned to the beginning of the Working-Storage Section if the program contains one. The rest are assigned to files in the order in which FDs occurred, and to additional Working-Storage.

If the PMAP, CLIST, or DMAP options are in effect, a list of permanently assigned registers and the BLs (base locators) associated with them is written on SYSPRINT.

At execution time, permanent base registers are loaded from the TGT by routine INIT3. (Registers 0 through 5 are work registers; instructions using these registers are generated from the Procedure A-text.) Registers 14 and 15 are used as temporary base registers.

To assign base registers in procedure instructions, phase 6 refers to and updates table REGMTX (internal to Phase 6) which contains an entry for each of registers 6 through 11, 14, and 15. Into an entry are

placed the BL type and BL number (the i and k of the idk field of an addressing parameter) of the area to which the register is currently pointing, and the status of the register (that is, how it is being used). When a field of the data area is the operand of a procedure instruction, table REGMTX is searched for a matching i and k. If it is found, this means that the register already contains the desired base locator and, therefore, the register can be used in the instruction.

If no register already contains the necessary base locator, an instruction is generated to load the base locator (which is stored in the TGT) into temporary base register 14 or 15.

When a register is used in an instruction, the status portion of the REGMTX entry is updated to indicate how it is currently being used. Status bits may also be updated by the A-text macro-type instruction elements (see Figure 45 in this chapter). A list of these elements and their meanings appears in the Procedure A-text formats in "Section 5. Data Areas."

PROCESSING DATA A-TEXT, E-TEXT, AND DEF-TEXT

The primary function of Data A-text processing is to place values into fields of the data area and Global Tables of the object program. Each element results in either the writing of an object text element or an entry in the RLDTBL. Some RLDTBL entries will later be written out as Relocation Dictionary (RLD-text) entries for the data area and as object text. Others, for the Global Tables, will be written as object text only. (These will be relocated by the object program.)

SYSUT4, from which Data A-text is read, also contains E-text generated by phases 10 through 51, and DEF-text for the cross-reference listing if the SXREF, XREF, VBREF, or VBSUM option is in effect. Figure 46 illustrates the contents of this data set when it is read by phase 6. Figure 47 describes how each type of element is processed.

PROCESSING THE RLDTBL TABLE

The maximum amount of space is obtained for the RLDTBL table by a call to GETALL in phase 00. The RLDTBL does not use a TIB. The calling sequence is:

```
L      15,=A(GETALL)
BALR   14,15
```

The starting address of the table area for the RLDTBL table is returned in register 0, and its length is returned in register 1.

After end-of-file has been reached on SYSUT4, the RLDTBL table is processed. First, indirect address constants are resolved.

The table is then sorted in ascending order of target address. Object text is written for items that are in the Global Tables. This text consists of address constant definitions that will be stored in the Global Tables at execution time. No RLD-text is required for these items, because the addresses are relocated during program execution by routine INIT3. Object text is also written for data area address constants (obtained from address constant and indirect address constant definitions). For the data area address constants, RLD-text is written so that the linkage editor can relocate the addresses.

INITIALIZATION CODING GENERATION

After the RLDTBL table has been processed, initialization coding is generated using the routines that process Procedure A-text.

There are three initialization routines, called INIT1, INIT2, and INIT3. All three are resident in the root segment if the program is segmented. INIT2 is generated first, followed by INIT3. In the generation of INIT3, the QTBL table is used to generate code that will call the Q-Routines at the beginning of program execution to initialize variable-length fields. INIT1 is generated last, because it contains pointers to INIT2 and INIT3. The contents and functions of these three routines are described in detail in "Appendix B. Object Module."

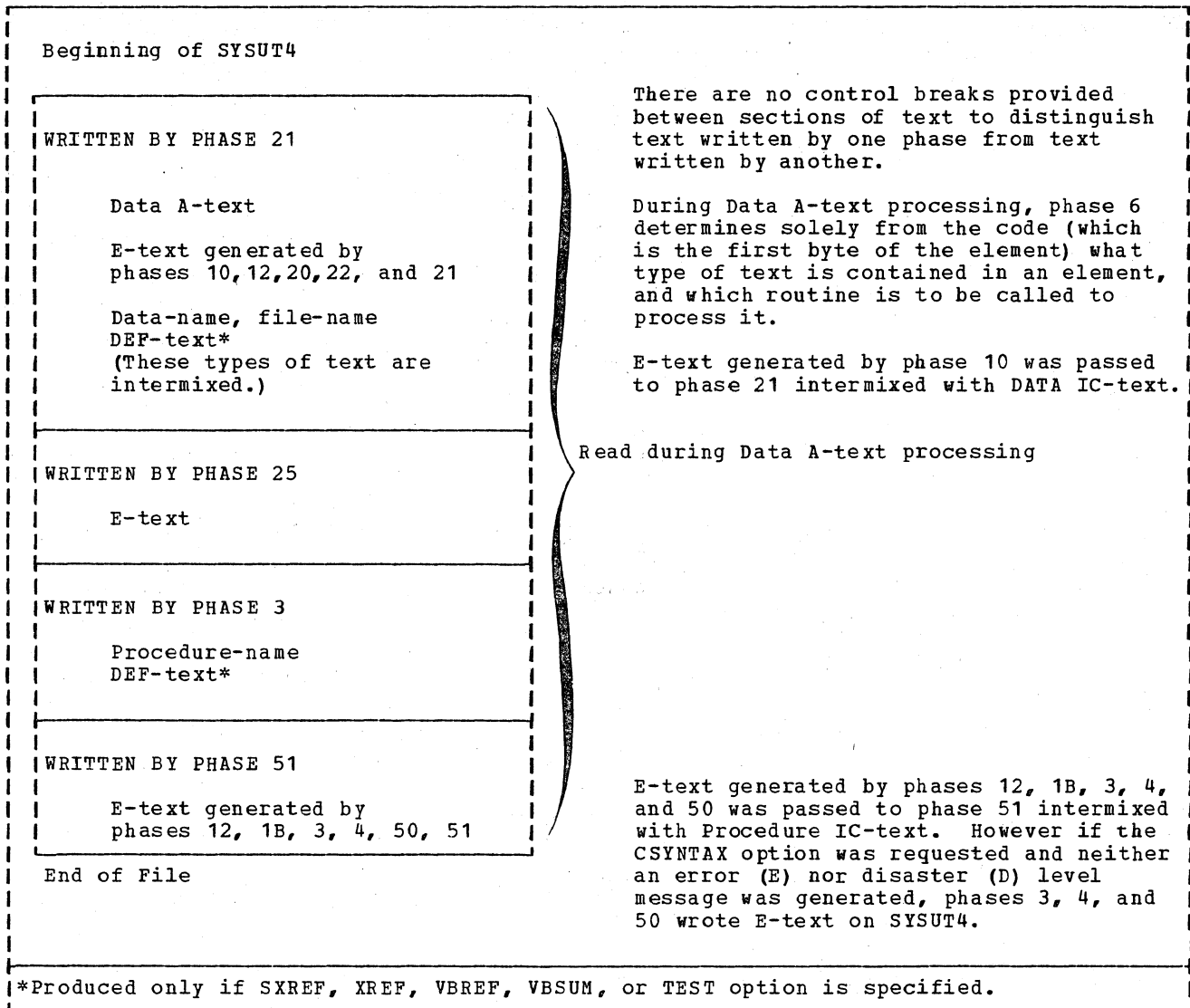


Figure 46. Contents of SYSUT4 When Read by Phase 6

Code and Type	Action Taken
00 E-text	<p>All E-text is built into a table called ERRTBL, which is passed to phase 70. Phase 6 does not process the E-text. If the ERRTBL table overflows the space allocated to it, all of the E-text is rewritten on SYSUT3, and a bit in the SWITCH field of COMMON is set to indicate this to phase 70.</p> <p>There are two types of E-text elements: message definitions and message parameters. Message parameters are optional; however, if they occur, one or more message parameters immediately follow the message definition to which they apply (the uses of these elements are explained in the chapter "Phases 70, 71, and 72"). Phase 6 examines each element to determine its length, so that the correct number of bytes may be stored in the table. To do so, it checks the third byte of the element. If the byte contains a zero, the element is a message definition whose length is eight bytes. If the third byte is nonzero, the element is a message parameter of variable length, and the length is determined from the value of the second byte (for the format of E-text and for the format of ERRTBL, see "Section 5. Data Areas").</p>
04 DCB address	<p>Generate an RLDTBL entry that will cause the address of the DCB to be placed in the correct cell of the DCBADR field in the PGT at execution time. Get displacement of the DCBADR field from cell DCBCTR in COMMON (see "DCBADR Allocation" in this chapter) and use the DCB number to compute displacement of cell. Text element contains the value (relative address of the DCB) to be placed in the PGT cell.</p>
08 DECB address	<p>Generate an RLDTBL entry that will cause the address of the DECB to be placed in the correct cell of the DECBADR field of the TGT at execution time. Get displacement of the DECBADR field from cell DECBCTR in COMMON (see Figure 36 in this chapter) and use the DECB number to compute displacement of cell. Text element contains the value (relative address of the DECB) to be placed in the cell.</p>
0C Block address	<p>Generate an RLDTBL entry that will cause the address of the buffer to be placed in the correct BL cell of the TGT at execution time. Get displacement of the BL field from cell BLCTR in COMMON (see Figure 36 in this chapter) and use the BL number to compute displacement of the cell. Text element contains the value (relative address of the buffer) to be placed in the TGT cell.</p> <p>If the value of the SIZE field of the element exceeds 1024 (SIZE specified length of the block in fullwords), more than one BL has been assigned to the buffer. For each 1024-word area after the first, another RLDTBL entry is made. The second RLDTBL entry will cause the buffer address plus 4096 to be placed in the next BL cell of the TGT.</p>
14 FIB address	<p>Generate an RLDTBL entry that will cause the address of the File Information Block (FIB) to be placed in the correct cell of the FIB field in the TGT at execution time. Get displacement of the FIB field from AMICTR cell in COMMON and use the FIB number to compute displacement of cell. Text element contains the value (relative address of the FIB) to be placed in the TGT cell.</p>
20 Data A-text	<p>Generate the COUNT option information.</p>

Figure 47. Processing Data A-text, E-text, and DEF-text (Part 1 of 2)

Code and Type	Action Taken
24 Working-Storage Section address	Generate an RLDTBL entry that will cause the address of the Working-Storage Section to be placed in the correct BL cell of the TGT at execution time. Get displacement of the BL field from cell BLCTR in COMMON (see Figure 36 in this chapter) and use the BL number to compute displacement of the item. Text element contains the value (relative address of the Working-Storage Section) to be placed in the TGT cell.  If the value of the SIZE field exceeds 1024 (SIZE specifies the length of the Working-Storage Section in fullwords), more than one BL has been assigned. For each 1024-word area after the first, another RLDTBL entry is made. The second entry will cause the address plus 4096 to be placed in the next BL cell.
28 Constant definition	Write object text that will place the value of the constant into a specified location in the data area at execution time. This type of element is used to fill some fields of DCBs and DECBs, and to initialize data items for which a VALUE clause was specified.
2C Address constant definition	Generate an RLDTBL entry that will cause the address of the procedure-name (PN) or generated procedure-name (GN) to be placed in a specified location of the data area at execution time. This type of element is used to place pointers to routines in DCBs and exit lists.
34 Q-Routine identification	This type of element contains a GN number for a Q-Routine. The elements are built into a table called QTBL. Each entry is resolved so that it contains the actual address of the routine rather than simply the GN number. This processing is identical to that for GN references in Procedure A-text (see Figure 45 in this chapter). When phase 6 generates the code of INIT3 (one of the execution-time initialization routines), it uses the QTBL table to generate a call to some Q-Routines to initialize the data and table areas affected by OCCURS...DEPENDING ON data items, where the object of the DEPENDING ON clause is an item in Working-Storage.
38 BL reference	Generate an RLDTBL entry that will cause the displacement in the TGT of the BL number assigned to VSAM files to be placed in a specified location of the data area at execution time.
3C BLL reference	Generate an RLDTBL entry that will cause the displacement in the TGT of the BLL numbers assigned to VSAM files in the Linkage Section to be placed in a specified location of the data area at execution time. This type of element is used to complete the building of the FIB at execution time.
48 Data-name or file-name DEF-text	This element is present only if the SXREF, XREF, VBREF, VBSUM or TEST option was specified. Each element is written out as it is encountered on SYSUT1, to be read by phase 6A. The chapter "Phase 6A" describes how these elements are used.
4C Procedure-name DEF-text	This element is present only if the SXREF, XREF, VBREF, VBSUM or TEST option was specified. Each element is written out as it is encountered on SYSUT1, to be read by phase 6A. The chapter "Phase 6A" describes how these elements are used.

Figure 47. Processing Data A-text, E-text, and DEF-text (Part 2 of 2)



PHASE 62

Phase 62 (IKFCBL62) is the first of the three phases that prepare a machine language program suitable for input to the linkage editor if the optimizer (OPT) option is specified. The elements of this program are described in the chapter "Object Module." The phase is divided into several sequential parts, each of which performs specific functions. The functions are:

- Determines object program storage allocation for the TGT (Task Global Table) by processing counters in COMMON and calculating the displacements of items which reside in the TGT at execution time.
- Optimizes literals and virtuals by processing Optimization A-text; determines storage allocation in the PGT (Program Global Table) for these items, for PN and GN cells, and for the DCBADR, VNI, and PROCEDURE BLOCK fields and calculates their displacements, using counters in COMMON.
- With Procedure A-text as input, determines approximate object program storage requirements for the Procedure Division by calculating the Procedure block number in which each PN or GN is located. If the program is segmented, groups the sections of instructions into segments.
- Optimizes usage for both permanent and temporary register assignments.

The operations of phase 62 are described in Diagram 5, located with the foldouts at the back of this publication.

OUTPUT OF PHASES 62, 63, AND 64

Copies of the object program and compilation information are put out by phases 62 and 64. Phase 64 may also put out REF-text and phase 63 may put out Debug-text. The output of phases 62, 63, and 64 depend on the compiler options specified by the user or determined by defaults set at installation time. The following are the options that determine the output produced:

<u>Option</u>	<u>Result</u>
PHAP	Causes the TGT, Literal Pool, PGT, register assignments, Working-Storage message, and a listing of the object text to be written on SYSPRINT.
CLIST	Causes the TGT, Literal Pool, PGT, register assignments, Working-Storage message, and a condensed object program listing to be written on SYSPRINT. The object program is limited to the card number, verb name (or verb number if the program is segmented), and address of the first instruction for each verb.
DMAP	Causes the TGT, Literal Pool, PGT, register assignments, and the Working-Storage message to be written on SYSPRINT. This option has already caused phase 3 to print a Data Division glossary.
XREF SXREF	Causes a source ordered (XREF) or alphabetically ordered (SXREF) cross-reference listing to be written on SYSPRINT. Phase 64 writes XREF-text for use by phase 6A.
LOAD	Causes the object program to be written on SYSLIN by phase 00.
DECK	Causes the object program to be written (punched) on SYSPUNCH by phase 00.
BATCH NAME	If both the BATCH and NAME options are specified, they cause a linkage editor control card to be generated at the end of the object program, so that the object program will be a separate load module. If BATCH is specified, the relative number of this compilation in the batch appears among the statistics printed by phase 64.
	<u>Note:</u> The linkage editor control card generated for the BATCH and NAME options is

produced by phase 65 if the FLOW, STATE, or SYMDMP option is specified.

- FLOW[=n[n]] Causes the flow trace facility to be included in the object program. The number [n[n]] of traces requested is retained in the FLOWSZ cell in COMMON and is passed to phase 65, which places the number in the variable portion of the TGT.
- STATE Causes the statement number facility to be included in the object program. Phase 63 writes Debug-text elements on SYSUT4 for use by phase 65.
- SYMDMP Causes the symbolic debug facility to be included in the object program. Phase 63 writes Debug-text on SYSUT4 for use by phase 65.
- OR
- TEST

For all compilations, compiler statistics are written by phase 64 on SYSPRINT from COMMON, where they were saved by phase 02.

If the PMAP, CLIST, or DMAP options have been specified, phase 62 causes the TGT, Literal Pool, PGT, register assignments, and Working-Storage message to be written on SYSPRINT. Phase 64 causes the object program listing for the PMAP or CLIST option to be written on SYSPRINT. If the FLOW, STATE, or SYMDMP option is in effect, phases 62 and 64 create the TGTADTBL table which is used by phase 65. Phases 62, 63, and 64 also use the TGTADTBL table for interphase communication before completing it in phase 64 for use by phase 65.

The user may specify both the LOAD and DECK options, in which case the object program is written on both SYSLIN and SYSPUNCH. He may also specify NOLOAD and NODECK; in this case, he receives no executable copy of his object program.

If no output was requested (no PMAP, LOAD, DECK, CLIST, DMAP, BATCH, NAME, SYMDMP, STATE, SXREF, XREF, VBREF, or VBSUM), text processing is bypassed unless the TERM option was specified. In this case, phase 64 scans the E-text on SYSUT4 and increments the ERRNUM cell in COMMON. After phase 64 scans the E-text, it rewinds SYSUT4 and returns to phase 00. Phase 00 uses the count in ERRNUM to write a message to SYSTEM giving the number of errors encountered for the compilation. Phase 64 also sets a bit in COMMON to indicate whether phase 70 is required (see "Suppression of Output Listing" below).

### Suppression of Output Listing

If the SUPMAP (suppress map) option is in effect, no output is produced by phases 62 and 64 if a D-level or E-level error message was generated by any phase. This is determined by testing the ERRSEV cell in COMMON. A value of 12 or greater means that at least one D-level or E-level message occurred.

The ERRSEV cell was set by phases 2, 3, 4, 50, and 51 every time they encountered or generated an element of E-text (see "E-text" in the chapter "Phase 51"). A test is made for this condition upon entering phase 62. If it occurs, a message is printed and the text is not processed unless SXREF or XREF is specified.

If the SUPMAP condition occurs and neither SXREF or XREF was requested, phase 62 terminates processing, calls phase 63, which does no processing and in turn calls phase 64, after phase 62 sets two bits in COMMON (bits 6 and 7 of the second byte of SWITCH). Bit 6 indicates that phase 70 is to be called, and bit 7 indicates to phase 70 that E-text must be read from SYSUT4. If the TERM option was specified, phase 64 scans the E-text on SYSUT4 and increments the ERRNUM cell in COMMON. After phase 64 scans the E-text, it rewinds SYSUT4 and returns to phase 00.

If SXREF or XREF was requested, text is processed by phases 62, 63, and 64 but the only output is REF-text and DEF-text (which phase 6A uses to produce the cross-reference listing). The ERRTBL is also built by phase 64, or E-text is written on SYSUT3. A bit is set in COMMON (bit 6 or the second byte of SWITCH), indicating to phase 6A that phase 70 is required. If E-text was written on SYSUT3, bit 5 of the second byte of SWITCH is set to indicate that E-text must be read from SYSUT3.

Phases 62 and 64 do not write object text in execution-time sequence. Rather they instruct the linkage editor to reorder the text by assigning relative addresses. To do this, they allocate space for areas that will be written later, altering the LOCCTR (location counter) cell of COMMON to reflect the relative location at execution time of the area currently being processed.

### TASK GLOBAL TABLE STORAGE ALLOCATION

When phase 62 receives control, the LOCCTR cell contains the relative address of the Task Global Table (TGT) in the load module.

LOCCTR was set by phases 22 and 21, which added the length of the data area to that of the INIT1 routine. (These areas precede the TGT in the load module.)

Routine TGTINT first does preliminary computations to determine the length of the entire TGT. If this length exceeds 4096 bytes, one 4-byte OVERFLOW cell is allocated for each 4096-byte area after the first. Then this routine computes the locations of TGT fields after the OVERFLOW cells.

Some fields of the TGT are constant in length; others are variable, depending on the requirements of the program being compiled. For most of the variable fields,

there is a counter in COMMON used to compute its length. When the value has been used, the counter is set to the displacement of the current field in the TGT. Figure 48 lists these counters and the TGT fields to which they correspond. In a register called RW1, a counter is kept of the displacement of the current field in the TGT.

Some of the counters in COMMON specify a number of bytes. Others specify a number of entries, where each entry requires two or four bytes. In the latter case, the value of the counter is multiplied by 2 or 4 before it is used to compute displacements.

Counter	Contents From Earlier Phases	TGT Field	Multiplication Factor
TSMAX	Number of bytes needed for arithmetic temporary storage.	TEMP STORAGE	8
TS2MAX	Number of bytes needed for nonarithmetic temporary storage.	TEMP STORAGE-2	1
TS3MAX	Number of bytes of work area for aligning non-SYNCHRONIZED data items.	TEMP STORAGE-3	1
TS4MAX	Number of bytes of work area for table-handling verbs.	TEMP STORAGE-4	1
BLLCTR	Number of base locators assigned to Linkage Section.	BLL	4
VLCCTR	Number of variable-length cells (containing current length of a variable-length field).	VLC	2
INDEX1	Number of index-names defined in INDEXED BY clause.	INDEX	4
SBLCTR	Number of secondary base locators (location of a field variably located because it follows a variable-length field).	SBL	4
BLCTR	Number of base locators assigned to files and Working-Storage.	BL	4
SUBCTR	Number of subscript save cells.	SUBADR	4
ONCTR	Number of ON control cells.	ONCTL	4
PFMCTR	Number of PERFORM control cells (for PERFORM X TIMES).	PFMCTL	4
*The number of VNs in the program is passed to phase 62 in the VNCTR cell of COMMON, not the VNLOC cell. However, this value is moved into the VNLOC cell and all further TGT processing uses VNLOC rather than VNCTR. (The number of VNs in the program must also be known for PGT allocation to determine the size of the VN field in the PGT. This value is saved in VNCTR.)			

Figure 48. Use of Counters in COMMON to Allocate Space in the TGT for Variable-length Fields (Part 1 of 2)

Counter	Contents From Earlier Phases	TGT Field	Multiplication Factor
PSVCTR	Number of PERFORM save cells.	PFMSAV	4
VNLOC*	Number of variable procedure-names.	VN	8
DECBCT	Number of DECBs.	DECBADR	4
XSWCTR	Number of EXHIBIT switches.	XSASW	1
XSACTR	Number of bytes for EXHIBIT saved area.	XSA	1
PARMAX	Area needed for parameter lists.	PARAM	4
RPTSAV	Report Writer save area requirements.	REPORT SAVE	4
CKPCTR	Number of checkpoint requests.	CHECKPT CTR	4
SA2CTR	USE LABEL or USE ERROR procedure save area requirements.	SAVE AREA-2	4
SA3CTR	Maximum number of files specified in an OPEN statement.	SAVE AREA-3	4
AMICTR	Number of FIBs for VSAM files.	FIB	4

Figure 48. Use of Counters in COMMON to Allocate Space in the TGT for Variable-length Fields (Part 2 of 2)

This is done in routine DSPLAC, which is called for each variable-length field. Two parameters are passed to this routine: the address of the counter in COMMON, and the number of bytes for each entry. From the number of bytes, DSPLAC also determines boundary alignments. DSPLAC places the value of RW1 (the displacement of the field in the TGT) into the counter, and adds the length of the field to RW1.

If the PMAP, CLIST, or DMAP options are in effect, DSPLAC calls routine MAPLOC, which prints one line at a time.

If the STATE, FLOW, SYMDMP, or TEST option is in effect, the SWITCH cell, the CURRENT PRIORITY cell (initialized to zero only), the DEBUG TABLE PTR cell, and the DEBUG TABLE information in the TGT are set by phase 65.

After the length of the entire TGT has been calculated, the value of RW1 (the length of the TGT) is added to the LOCCTR cell. The value of the LOCCTR cell is now the displacement of the PGT.

OPTIMIZING STORAGE FOR THE PROGRAM GLOBAL TABLE

The general function of this part of phase 62 is to allocate space for the Program Global Table (PGT) in the same way that TGT storage was allocated. Before this can be done, however, the required space must be determined for the literals, virtuals, and procedure-names that reside in this table at execution time. The routines that determine the lengths of these fields also optimize the contents of the fields by eliminating duplication.

For optimizing, phase 62 reads Optimization A-text from SYSUT3 and merges its information with information from tables and counters generated by earlier phases. This information is used to perform the following functions:

- Process virtual reference definitions for library subroutines to be called at execution time.
- Build the VN priority (VNPTY) table.
- Optimize and calculate storage requirements for literals, DISPLAY literals, and virtuals.

- Build the VNPNTBL, BLVNTBL, PNATBL, and GNATBL tables.

Building the VN Priority Table

VN definition elements are not used for optimization. They are included in the Optimization A-text for a segmented program because they are used to build a table (called VNPTY) which must be in storage for phase 62 and 63 processing. As an element is read, it is entered unchanged into this table. After the Optimization A-text data set has been closed, the VNPTY table is sorted in ascending order of priority number.

Processing PNs and GNs

Phase 62 builds the VNPNTBL, BLVNTBL, PNATBL, and GNATBL tables for PN and GN processing for the PGT. The VNPNSORT routine builds the VNPNTBL table from the VN EQUATE PN or VN EQUATE GN elements of Optimization A-text.

The GNVNRTN routine builds the BLVNTBL table from GN-VN PERFORM elements.

The PGNARTN routine builds the PNATBL and GNATBL tables. These tables list the PNs and GNs for which address constant cells are required in the PGT.

Optimizing Literals and Processing DISPLAY Literals

The literal optimization routines are used to eliminate storage duplication in cases where the source programmer used the same literal more than once. Routine LTLRTN processes internal literals, and routine LTLDIS processes DISPLAY literals. These routines build three tables: the CONTBL and CONDIS (for regular and DISPLAY literals, respectively) tables contain one entry for each unique literal, and the LTLTBL table contains an entry for each use of a literal.

DISPLAY literals are entered into the CONDIS table.

When a nonliteral literal definition is encountered, the CONTBL table is searched for an entry identical to the literal. (To be identical, two internal literals must meet the same boundary requirements as well as have the same value.) If no match is

found, the new literal is entered into the CONTBL table. Any bytes skipped because of boundary alignment are filled with zeros. The displacement of this entry from the beginning of the table is placed in the LTLTBL table, with a bit set to indicate whether it is a CONTBL or CONDIS entry. If a match is found, only an LTLTBL entry is made. This LTLTBL entry is the displacement of the CONTBL entry that matched the literal being processed.

Figure 49 shows an example of these tables after all Optimization A-text has been processed. The Optimization A-text contained literal definition elements for the following literals:

8, 3(DISPLAY), 3, 9, Y(DISPLAY), 8

After the Optimization A-text data set is closed, the Literal Pool is written on SYSPRINT using the contents of the CONTBL and CONDIS tables, if the PMAP, CLIST, or DMAP options are in effect. The Literal Pool is also written on SYSPUNCH and SYSLIN, as needed.

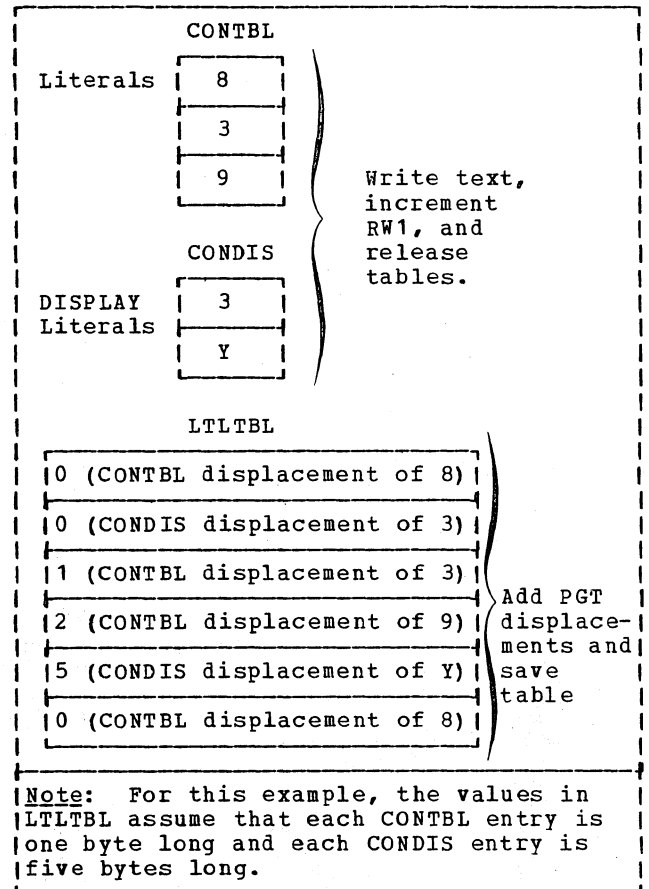


Figure 49. CONTBL, CONDIS, and LTLTBL Tables after Processing Literals

Optimizing Virtuals

The virtual optimization routine (VIRRTN) is used to eliminate storage duplication in cases where the same EBCDIC name of a called program is referred to in more than one CALL statement or in more than one call to a COBOL library object-time subroutine. The logic of this processing is similar to that of literal optimization. Two tables are built: the CVIRTB table contains one entry for each unique virtual, and the VIRPTR table contains one entry for each reference to a virtual.

When a virtual definition is encountered, the CVIRTB table is searched for an identical entry. If none is found, the new virtual is entered in the CVIRTB table. Into the VIRPTR table is entered the displacement of this virtual from the beginning of the CVIRTB table. If a match is found, only a VIRPTR entry is made.

This VIRPTR entry contains the displacement in the CVIRTB table of the entry that matched the virtual being processed.

Figure 50 shows the contents of these tables after processing Optimization A-text for a program containing the following virtuals:

CVIRT1, CVIRT2, CVIRT3, CVIRT1, CVIRT2.

ALLOCATING STORAGE FOR THE PROGRAM GLOBAL TABLE

When all Optimization A-text has been read, storage is allocated for the PGT. Entries for the External Symbol Dictionary and object text are generated for virtuals and literals. Register RW1 is used throughout PGT allocation to hold the displacement of the field currently being processed. Counters in COMMON are set to the displacements of their corresponding PGT fields from the beginning of the PGT.

If the PMAP, CLIST, or DMAP options are in effect, the format of the PGT is written on SYSPRINT using routine MAPLOC.

DEBUG LINKAGE AREA Allocation

If the SYNDMP option is in effect, 12 bytes are allocated for the DEBUG LINKAGE AREA. Otherwise, no space is allocated for the field in the PGT.

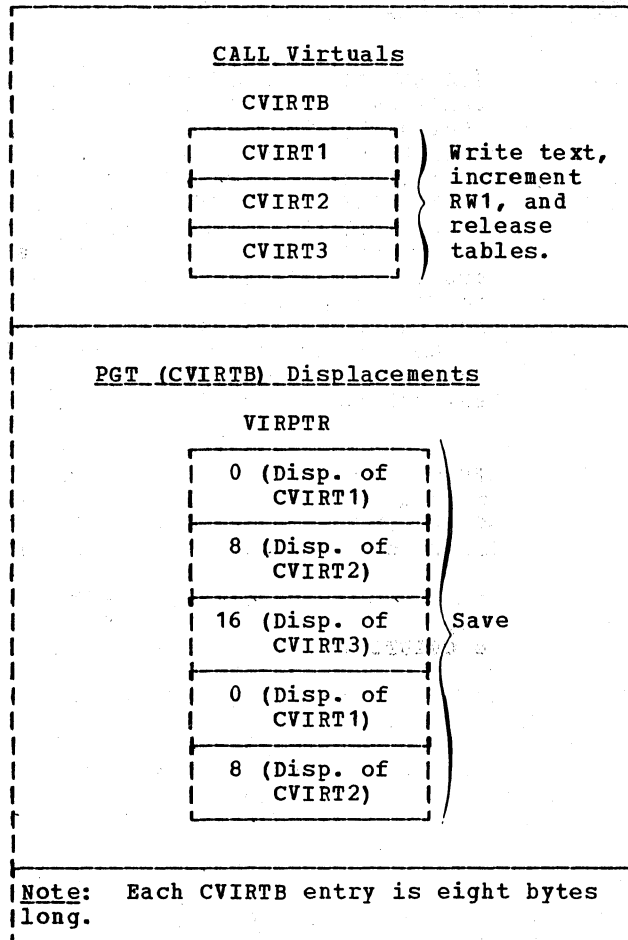


Figure 50. CVIRTB and VIRPTR Tables after Processing Virtuals

OVERFLOW Allocation

Preliminary calculations are made to determine whether the size of the PGT exceeds 4096 bytes. If it does, one 4-byte OVERFLOW cell is required for each 4096-byte area after the first. Since OVERFLOW cell allocation occurs before the first reading of Procedure A-text in this phase, phase 62 cannot yet determine the number of PROCEDURE BLOCK cells that are required in the PGT. Therefore, it allocates one additional OVERFLOW cell to allow for the possibility that allocation of the PROCEDURE BLOCK cells may cause the PGT to exceed the final 4096-byte area that has already been allocated.

VIRTUAL Allocation

After the OVERFLOW CELLS field of the PGT has been calculated, the VIRTUAL field is processed. Using the CVIRTB table, an External Symbol Dictionary entry (ESD-text type 2) is written for each virtual unless the DYNAM or the RESIDENT option is in effect. (If the RESIDENT option is in effect, no ESD or RLD item is written for a library subroutine; if the DYNAM option is in effect, no ESD or RLD item is written for a library subroutine or a user subprogram.) Object text is also written and entries are made in the RLDTBL table for subsequent writing of the Relocation Dictionary.

The VIRCTR cell of COMMON is set to the displacement of the VIRTUAL field from the beginning of the PGT. (This value is 0 unless OVERFLOW cells have been allocated.)

To determine the length of the VIRTUAL field, four bytes are allowed for each entry in the CVIRTB table. The calculated length is added to register RW1. If the DYNAM and/or RESIDENT option is in effect, the CVIRTB table is used to enter in the PGT the EBCDIC names of those routines that are to be dynamically loaded. The CVIRTB table is kept for use during Procedure A-text processing, when its contents (EBCDIC names) will be used to generate comments for CALL statements.

The entries in the VIRPTR table are changed to contain displacements in the VIRTUAL field (see the example in Figure 51. The table is saved for subsequent use during Procedure A-text and Procedure A1-text processing.

VIRTUAL EBCDIC NAMES Allocation

If the DYNAM or RESIDENT option is in effect, an 8-byte cell for each library subroutine name is allocated in the PGT; in addition, if the DYNAM option is in effect, an 8-byte cell for each user subprogram name is allocated in the PGT. If neither option is in effect, this field does not exist.

A count of the EBCDIC names to be placed in the PGT is kept in the BCDCTR cell in COMMON. This count is multiplied by 8 to reserve space for the list of names. After VIRTUAL allocation, register RW1 contained the displacement of the VIRTUAL EBCDIC NAMES field; the displacement is saved in the BCDISP cell in COMMON and register RW1 is incremented to reflect the allocated bytes for the VIRTUAL EBCDIC NAMES cells.

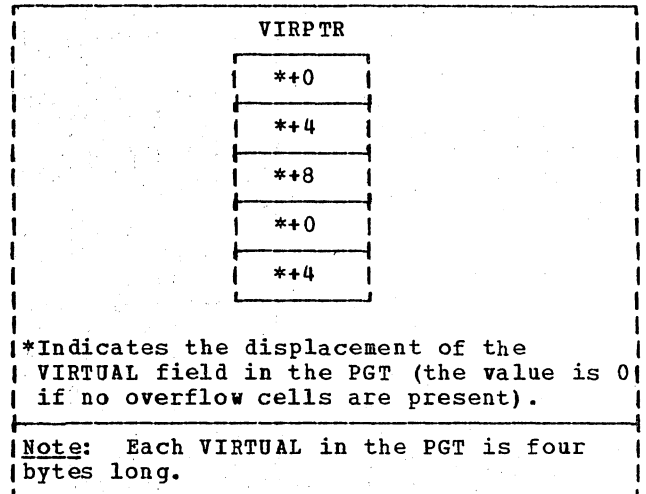


Figure 51. VIRPTR Table after VIRTUAL Allocation

PN Allocation

After the VIRTUAL field or the VIRTUAL EBCDIC NAMES field, if allocated, has been processed, the value in register RW1 is the displacement of the PN field in the PGT. This value is saved in the PNCTR cell in COMMON.

Only those PNs that follow TO PROCEED TO in an ALTER statement and the section-names defined in a USE statement in the Declaratives Section require PN cells in the PGT. Phase 51 sets the RPNCTR counter in COMMON to the number of cells required. Phase 62 uses the RPNCTR counter to allocate 4 bytes for each PN. The total length of the PN field is then added to register RW1.

GN Allocation

After the PN field has been processed, the value in register RW1 is the displacement of the GN field in the PGT. This value is saved in the GNCTR cell in COMMON.

Only those GNs that are used in instructions for an AT END phrase or an INVALID KEY option require GN cells in the PGT. Phase 51 sets the RGNCTR counter in COMMON to the number of cells required. Phase 62 uses the RGNCTR counter to allocate 4 bytes for each GN. The total length of the GN field is then added to register RW1.

DCBADR Allocation

The DCBCTR cell in COMMON is used to determine the amount of space required: 4 bytes are reserved for each DCB in the program. The DCBCTR cell is set to the displacement of the DCBADR field (the value of register RW1), and RW1 is then incremented to reflect the allocated bytes.

VNI Allocation

The VNCTR cell of COMMON was used by earlier phases to count the number of variable procedure-names in the program. Eight bytes are allocated for every VN. The displacement of the VNI field (the value of RW1) is placed in the VNILOC cell, and the number of bytes allocated is added to register RW1.

LITERAL Allocation

The displacement of the LITERAL field in the PGT (the current value of register RW1) is placed in the LTLCTR cell of COMMON. The length and contents of the LITERAL field will be identical to the CONTBL and CONDIS tables.

For each LTLTBL entry that refers to CONTBL, the value in the LTLTBL table is replaced by the displacement of the specific literal from the beginning of the PGT. This displacement is calculated by adding the value already in the LTLTBL entry (which is the CONTBL displacement of the literal) to the value of RW1. An example is shown in Figure 52.

The same processing occurs for LTLTBL entries that refer to CONDIS, except that the increment includes the length of the CONTBL table. This occurs because DISPLAY literals are placed after internal literals in the PGT, as illustrated by Figure 38. The LTLTBL table is saved for use during Procedure A-text processing. The lengths of the CONTBL and CONDIS tables are used to increment RW1. The contents of the tables are used to write object text, and the tables are released.

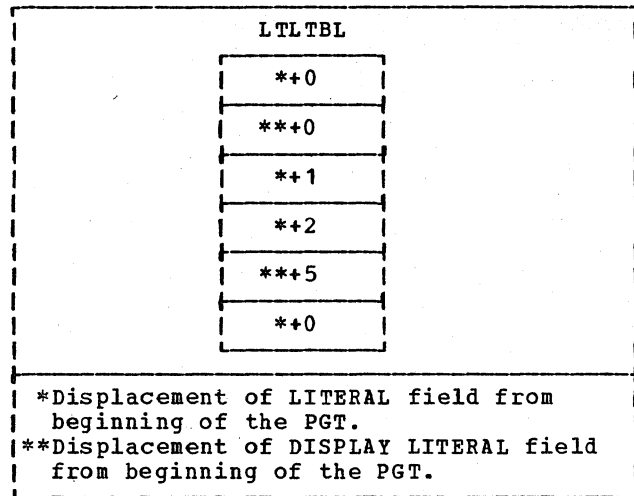


Figure 52. LTLTBL Table after Literal Allocation

PROCEDURE BLOCK Allocation

Phase 62 does not allocate storage for the PROCEDURE BLOCK field until after it reads and processes Procedure A-text. It reads Procedure A-text to determine the number of blocks, containing approximately 4096 bytes of storage, that are required for the optimized Procedure Division (see "Optimizing and Allocating Storage for the Procedure Division" later in this chapter).

After the allocation of storage for the other fields of the PGT, the value in register RW1 is the displacement of the PROCEDURE BLOCK field in the PGT. This value is saved in the PRBLDISP cell in COMMON. After Procedure A-text processing, phase 62, using the PROCBL counter, allocates one 4-byte cell for each Procedure block.

OPTIMIZING REGISTER ASSIGNMENTS

Seven registers are used by the compiler to address Data Division items or OVERFLOW cells in the machine language program. Registers 6 through 10 are assigned permanently, that is, for the entire object program; registers 14 and 15 are assigned on a temporary basis, that is, for single instructions or for short sections of code only. Phase 62 assigns registers 6 through 9 before Procedure A-text is read, and register 10 after Procedure A-text is read. Use of registers 14 and 15 is determined as Procedure A-text is processed. Optimization takes place for both permanent and temporary register assignments.



#### PERMANENT REGISTER ASSIGNMENTS

Phase 62 builds the BLASGTBL table for permanent register assignments. Before Procedure A-text is read, the REGMV1 routine assigns the OVERFLOW cells of the TGT and the PGT to permanent registers, starting with register 6, except for the OVERFLOW CELL. Register 12 points to the PGT permanent register. Next, the BLSRCH routine searches the BLUSTBL table, built by phases 50 and 51, to determine the most frequently used base locator. That base locator is assigned to the next unassigned register. The process is repeated until each of registers 6 through 9 has been assigned. After Procedure A-text has been read, register 10 is assigned to the next most frequently used base locator if it is not needed for the additional OVERFLOW cell of the PGT.

#### TEMPORARY REGISTER ASSIGNMENTS

Registers 14 and 15 are assigned to base locators for single instructions or for short blocks of object code only. While Procedure A-text is being read, the ENTDRP and ENTDRPL routines build the DRPTBL and DRPLTBL tables, respectively, to optimize the assignment of these registers.

Phase 62 optimizes the assignment of temporary registers by avoiding unnecessary repetition of load instructions. To do this, it assigns the first two unique base locators referred to in Procedure A-text to

registers 14 and 15 by making entries in the DRPLTBL table. It builds the DRPTBL table from the subsequent base locators referred to until a condition is met which makes resolution of register assignment possible. These conditions are as follows:

- A reference to a base locator whose previous assignment to a register is still in effect.
- A PN, a GN, or the entry point of a new program segment.
- The base locator that will be permanently assigned to register 10 if that register is not needed for the extra OVERFLOW cell of the PGT.
- A RESERVE, DESTROY, FREE, or BLCHNG element.

Figure 53 exemplifies the optimizing process for base locator assignments to register 14 and 15. Phase 62 builds the DRPLTBL table for address increment elements as well as for address references. DRPLTBL table processing for address increments is done only if the increment is greater than 4095, in which case an additional generated instruction is needed to load the address of the data-name plus the address increment, which is at least 4096 bytes, into temporary register 14 or 15. Phase 62 adds 4 bytes to ACCUMCTR for each such LA instruction needed. The DRPLTBL entry for an address increment indicates to phase 63 which temporary register to use in the RX field of the LA instruction being generated.

Steps	Temporary BL's	DRPTBL	Steps	DRPLTBL
①	BL=6			R 14*
②	BL=7			R 15*
③	BL=8	> 8	⑤	R 15*
④	BL=10	> 10	⑤	R 15*
⑤	BL=6			R 14**
⑥	BL=10			R 15**
⑦	BL=11	> 11	⑨	R 14*
⑧	BL=12	> 12	⑨	R 14*
⑨	BL=5***			R 14*
⑩	BL=6			R 14*
⑪	BL=10			R 15*
⑫	BL=7	> 7	⑬	R 14*
⑬	BL=10			R 15**
⑭	BL=12	> 12	⑮	R 14*
⑮	Entry point (referenced PN or GN definition or new segment)			

\*Indicates that a load instruction is generated for this assignment by phase 63.

\*\*Indicates that the BL is already loaded and that no load instruction is generated by phase 63.

\*\*\*BL=5 is the base locator that will be loaded into register 10 if the additional OVERFLOW CELL for the PGT is not required.

Explanation:

The first two base locators which are read, BL=6 and BL=7 (steps ① and ②), are entered immediately into the DRPLTBL table. This indicates to phase 63 that instructions are to be generated to load these base locators into registers 14 and 15, respectively.

The next two base locators that are read, BL=8 and BL=10 (steps ③ and ④), are entered into the DRPTBL table for later assignment to the DRPTBL table for later assignment to the DRPLTBL table.

Since the next base locator, BL=6, has already been assigned to register 14, it is assigned again (step ⑤) to register 14 in the DRPLTBL table and a flag is set to indicate that no load instruction is necessary.

BL=8 and BL=10 are then assigned to register 15 (step ⑤).

BL=10 (step ⑥) is assigned to register 15 and a flag is set to indicate that no load instruction is necessary.

BL=11 and BL=12 are entered into the DRPTBL (steps ⑦ and ⑧).

Since BL=5 is to be loaded into register 10 if it becomes available after Procedure A-text is read, it is conditionally assigned to register 14 and BL=11 and BL=12 are also assigned to register 14 (step ⑨).

The entire process is essentially repeated (steps ⑩ through ⑭) until the entry point of a procedure or segment is met (step ⑮) where the process is initiated again.

Figure 53. Optimizing Assignment of Registers 14 and 15

OPTIMIZING AND ALLOCATING STORAGE FOR THE PROCEDURE DIVISION

Whenever a PN or GN is referred to in an instruction, a check is made to determine whether the address of the Procedure block that contains the PN or GN has already been loaded into register 11. If it has not been loaded, then an instruction is generated to load the address of the Procedure block into register 11.

As phase 62 reads Procedure A-text, it determines the Procedure block number for each PN and GN and builds the PNLABTBL and GNLABTBL tables. These tables are passed to phase 63, which generates the actual instructions necessary for establishing addressability.

To build the PNLABTBL and GNLABTBL tables phase 62 uses a counter, called ACCUMCTR, to generate displacements within Procedure blocks. This counter is incremented with the length of each instruction occurring in the completed object program. Phase 62 uses ACCUMCTR to determine when the displacement of the definition of a GN or PN from the beginning of the Procedure block exceeds 4095 bytes. When the displacement is greater than 4095 bytes, a new Procedure block is begun.

Phase 62 also builds the PNFWDDBT and GNFWDDBT tables for all PNs and GNs that are referred to prior to their definition point. Since it cannot be determined whether the reference and the definition occur within the same Procedure block, it is not possible to determine whether the Procedure block address of the definition is already loaded into register 11 at the point where the reference is made to it. The forward branch tables (PNFWDDBT and GNFWDDBT) are used to accumulate the number of forward references to PNs and GNs, respectively, which may or may not be defined in a different Procedure block. As the definition point for a PN or GN that has been entered into the PNFWDDBT or GNFWDDBT table is encountered, the counter for that PN or GN is set to zero.

Since references to PNs and GNs that have not yet been defined may entail an additional instruction to load the Procedure block address of a different Procedure block, the number of bytes represented by the counters in the PNFWDDBT and GNFWDDBT tables must be added to the displacement in ACCUMCTR to determine the current length of the Procedure block.

Building the PNLABTBL and GNLABTBL Tables

Phase 62 sets a counter, called PROCBL, for use in building the PNLABTBL and GNLABTBL tables. PROCBL is incremented for each procedure block. The value contained in PROCBL is the Procedure block number for the current block of code. Using PROCBL, the DEF11 routine enters the Procedure block number of each referenced PN and GN definition in the PNLABTBL and GNLABTBL tables for use by phase 63.

The NOBLST routine uses the PNCTR and GNCTR cells in COMMON to determine the number of PN entries and GN entries, respectively, that are required in the PNLABTBL and GNLABTBL tables.

Incrementing the ACCUMCTR Counter

As phase 62 reads Procedure A-text, it increments ACCUMCTR by the length of each machine language instruction that is part of the completed object program. Since the optimizer phases of the compiler use Procedure block addresses to address PNs and GNs, these phases eliminate and change some of the instructions in Procedure A-text. Phase 62, therefore, determines which instructions are to be eliminated or changed during the optimization process, and then increments ACCUMCTR accordingly. To increment ACCUMCTR, phase 62 uses the codes listed in Figure 54, Procedure A-text, the PNLABTBL, GNLABTBL, PNFWDDBT, and GNFWDDBT tables, and the PROCBL counter.

Code	Meaning/Procedure-Name Definition	Action Taken By		
		Phase 62	Phase 63	Phase 64
C001	Load instruction not followed by branch instruction.			
	A Procedure-name was defined in same Procedure block.	Add 4 to ACCUMCTR.	Set C001 switch; replace L instruction with LA instruction. Add Procedure base register element. <sup>2</sup> Add 4 to counters. Do not rewrite C001.	Fill in displacement using PNLBDTBL or GNLBDTBL table and Procedure base register element. <sup>2</sup>
	B Procedure-name was defined in different Procedure block.	Add 8 to ACCUMCTR.	Set C001 switch; generate: L R11, Procedure block number element. <sup>3</sup> Generate LA instruction. Add Procedure base register element. <sup>2</sup> Add 8 to counters. Do not rewrite C001.	Fill in displacement of Procedure block in PGT using Procedure block number element. <sup>2</sup> Fill in displacement of procedure-name using PNLBDTBL or GNLBDTBL table and Procedure base register element. <sup>2</sup>
	C Procedure-name is not yet defined (forward reference).	Enter procedure-name in PNFWDDBTB or GNFWDDBTB table. Resolve procedure-name definition status by end of Procedure block.		

**Notes:**

<sup>1</sup>These Phase 50 optimization information elements (C0xx) are created by phase 50 from Phase 40 optimization information elements (43xx).

<sup>2</sup>Procedure Base Register Element:

Bytes	0	1	2-3
	C8 = PN CC = GN	Register number	PN/GN number

<sup>3</sup>Procedure Block Number Element:

Bytes	0	1
	C4	Block number

Figure 54. Processing for Optimization Information Elements (Part 1 of 3)

Code	Meaning/Procedure-Name Definition Status	Action Taken By								
		Phase 62	Phase 63	Phase 64						
C002	Branch-in point. (Addressability for Procedure block is uncertain.)	Add 4 to ACCUMCTR.	Indicate that Procedure block address is to be loaded at next reference to PN or GN. Do not rewrite C002.							
C003	An address constant is to be used for this element; PGT to contain a PN cell or GN cell.	Add 4 to ACCUMCTR.	Write Procedure A1- text element identical to Procedure A-text element. Add 4 to counters. Do not rewrite C003.	Process PN or GN reference as in phase 6.						
C004	PERFORM exit.	Find all entries in the BLVNTBL for the VN whose reference follows this element. Enter current block number in these table entries.	Do not rewrite C004.							
C005	Return point from a performed procedure (GN definition). Element is followed by a GN definition element.	Add 4 to ACCUMCTR.	Search BLVNTBL to determine if the EXIT from the performed procedure is in the same Procedure block as the return point. Rewrite GN definition element without C005. If PERFORM Exit and return point are in same block, rewrite element without C005. If they are not in same block, indicate that register 11 contains Procedure block address of PERFORM exit. Rewrite element without C005.	Fill in displacement of Procedure block in PGT, using Procedure block number element. <sup>3</sup>						
<p><b>Notes:</b></p> <p><sup>1</sup>These Phase 50 optimization information elements (C0xx) are created by phase 50 from Phase 4 optimization information elements (43xx).</p> <p><sup>3</sup>Procedure Block Number Element:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">Bytes</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> </tr> <tr> <td></td> <td style="text-align: center;">C4</td> <td style="text-align: center;">Block number</td> </tr> </table>					Bytes	0	1		C4	Block number
Bytes	0	1								
	C4	Block number								

Figure 54. Processing for Optimization Information Elements<sup>1</sup> (Part 2 of 3)

Code	Meaning/Procedure-Name Definition Status	Action Taken By		
		Phase 62	Phase 63	Phase 64
C006	Load instruction followed by an unconditional branch.			
A	Procedure-name defined in same Procedure block.	Add 4 to ACCUMCTR for RX-type branch instruction to be generated.	Turn on LOADSW switch. Do not rewrite C006.	
B	Procedure-name defined in different procedure block	Add 8 to ACCUMCTR for load of register 11 and RX-type branch instruction to be generated.	Turn on LOADSW switch. Do not rewrite C006.	Fill in displacement of Procedure blocks in PGT, using Procedure block number element. <sup>3</sup>
C	Procedure-name not yet defined.	Enter procedure-name in PNFWDBTB or GNFWDBTB; resolve procedure-name definition status by end of Procedure block.	Turn on LOADSW switch. Do not rewrite C006.	Write RX-type branch instruction following the C006 load instruction.

**Notes:**

<sup>1</sup>These Phase 50 optimization information elements (C0xx) are created by phase 50 from Phase 4 optimization information elements (43xx).

<sup>3</sup>Procedure Block Number Element:

Bytes	0	1
	C4	Block number

Figure 54. Processing for Optimization Information Elements<sup>1</sup> (Part 3 of 3)

**PROCESSING FOR BRANCH INSTRUCTIONS:** The PROCBL counter contains the number of Procedure blocks that are required for the Procedure Division. Each time that ACCUMCTR and the information in the PNFWDBTB and GNFWDBTB tables indicate that a PN or GN definition is at a location greater than 4095 bytes from the start of the Procedure block, block transition takes place. The PROCBL counter is incremented, and ACCUMCTR is set to zero.

When a branch is taken to a PN or GN within the Procedure block whose address already is loaded into register 11, an addition of 4 bytes is made to ACCUMCTR for the RX-type branch instruction. When a branch is taken to a PN or GN whose Procedure block is not already loaded into register 11, 8 bytes are added to the ACCUMCTR for the load of the Procedure block address and the RX-type branch instruction.

Each time that the ENTPT01 routine processes a PN or GN definition, it determines whether the value in ACCUMCTR plus the number of bytes necessary to branch to the procedure-names listed in the PNFWDBTB or GNFWDBTB table is greater than 4095 bytes. If it is not greater, then Procedure A-text processing continues. If it is, then the ENTPT01 routine determines whether the definition being processed has a count of forward references in the PNFWDBTB or GNFWDBTB table. If it does not, a new Procedure block begins at this definition point. If a count is found, however, the number of bytes represented by the count is compared with the number of bytes in ACCUMCTR minus 4096. If the count value is low or equal, a new Procedure block begins at this definition point. If it is high, the count field is set to zeros and this definition point remains within the current block. Phase 62 then makes new calculations to determine the size of the Procedure block.

PHASE 63

Phase 63 (IKFCBL63) is the second of the three phases that produce the machine language program. Its principal function is to produce Procedure A1-text, which is written on SYSUT2. Phase 63 produces the text according to the information supplied from phase 62. Upon completion, the text is passed to phase 64 where it is used to write the machine language program. Phase 63 also produces Debug-text on SYSUT4 if the STATE or SYMDMP option is in effect.

Phase 63 produces Procedure A1-text from Procedure A-text by:

- Inserting information for addressing PNs and GNs and Procedure blocks in instructions, such as displacements of PNs and GNs within a given block and the Procedure block number to be used.
- Generating all remaining instructions for the object program except the load instruction elements required when a data-name is only temporarily addressable.
- Reading the program in ascending order and writing Procedure A1-text in this order with the root segment first.

The operations of Phase 63 are described in Diagram 6, located with the foldouts at the back of this publication.

INITIALIZATION OF PHASE 63

Routine PHAS63 performs the initialization process for phase 63. It saves LOCCTR for restoration at end of file, relocates all of the TIB addresses, and primes all the new tables used by the phase except for the QGNTBL, which is primed in routine QBEGIN if there are Q-Routines. If the program is segmented, a call to phase 00 is issued to request a POINT to the first section of text on SYSUT1. Otherwise, the initialization routine requests phase 00 to read the first Procedure A-text buffer from SYSUT1.

CONSTRUCTING PROCEDURE A1-TEXT

Phase 63 reads Procedure A-text from SYSUT1 and writes Procedure A1-text on SYSUT2.

Procedure A1-text is described in "Section 5. Data Areas."

CONTROL ROUTINE

Routine GET serves as the control routine for phase 63 processing of Procedure A-text elements. It reads each element of Procedure A-text and branches to one of several routines for specific processing of each type of element.

For Phase 50 and 51 optimization information (C0) elements, macro-type instruction (44) elements, and operation code (48) elements, it branches to routines C0, MACRO and FOURTY8, respectively.

For each of the other elements the GET routine uses the GETBTBL table to branch to the proper routine for specific processing of that element.

PROCESSING PROGRAMS WITH ONE PROCEDURE BLOCK

In programs that do not exceed one Procedure block in length, and that have no Report Writer or Declaratives Section, the Procedure block address is loaded into register 11 only when the ENTRY macro-type instruction (4404) element and/or START macro-type instruction (4420) element of Procedure A-text is read.

PROCESSING FOR BRANCH INSTRUCTIONS

The BRANCH routine processes the branch element that follows the PN or GN reference. It uses the SAVETBL and either the PNLABEL or GNLABTBL table to determine whether the referenced PN or GN is defined in the Procedure block currently loaded in register 11. If the PN or GN is defined outside of the Procedure block, a Procedure A1-text element is generated to load register 11 with the address of the Procedure block that does contain the PN or GN definition. The BRANCH routine then generates an RX-type Procedure A1-text branch element instead of the RR-type Procedure-A text element so that an instruction will be generated to branch to the PN or GN.

If the PN or GN is defined within the Procedure block that is currently loaded in register 11, the routine merely changes the RR-type branch instruction to an RX-type branch instruction. The register number in the original instruction is changed to zero since register 11 is inserted in the branch instruction by phase 64. The PN (C8) or GN (CC) number element from the SAVETBL work area follows the instruction.

#### PROCESSING FOR OPTIMIZATION INFORMATION ELEMENTS (C001-C007)

Phase 63 processing for optimization information elements is described in Figure 54 in the chapter "Phase 62."

#### PROCESSING FOR RPT-ORIGIN (D4) ELEMENT

An RPT-ORIGIN (D4) element indicates that an RLDTBL table entry is to be made for this location in the program. The location is the point of definition (or the point of definition plus 4 bytes) of the GN for a REPORT-ORIGIN verb. Routine D4 creates the RLDTBL entry for the location, saving the value contained in LOCCTR. It sets the high-order byte to hexadecimal '10' to indicate the purpose of this entry to phase 64. The entry is used to generate text cards at the proper location; but phase 64 does not produce an RLD card for this type of entry. When the ORIGIN macro-type instruction (4438) element is read later, LOCCTR is set by phase 64 to the value of LOCCTR at the time of the RPT-ORIGIN element.

#### PROCESSING FOR ADDRESS REFERENCE (78) ELEMENTS

Address reference (78) elements are generated to address data areas. The data areas are addressed by means of a displacement from a base locator.

When an Address reference (78) element is found in Procedure A-text, routine GET branches to the ADREF routine for processing.

#### PROCESSING FOR ADDRESS INCREMENT (80) ELEMENTS

When Address increment (80) elements occur, they follow Address reference (78) elements

and indicate that an additional displacement value is to be added to the value indicated by the Address reference (78) element to address a data-name.

When routine GET finds an Address increment (80) element in Procedure A-text, it branches to routine ADINCR which determines whether the sum of the displacement and the value contained in the Address reference (78) element is less than 4096 bytes. If the sum is less, then the ADINCR routine adds a byte containing X'00' to the Address reference (78) element and writes the element in Procedure A1-text.

If the ADINCR routine determines that the sum is 4096 bytes or greater, it adds a byte containing X'0E' or X'0F', which indicates to phase 64 that IA instructions [that is, LA 14,4095(R) or LA 15,4095(R)] are to be generated using either register 14 or register 15, respectively, to the Address increment (80) element and writes the element in Procedure A1-text. LOCCTR and ACMCTR are incremented by 4 for each multiple of 4095 bytes in the added displacement.

#### PROCESSING FOR INCREMENTED ADDRESS (A4) ELEMENTS

The Incremented address (A4) element in Procedure A-text is functionally a combination of the Address reference (78) and Address increment (80) elements and is treated as such by phase 63. The ADREF routine (if no load is required, that is, if the data-name is already in a register) changes the element into a Base displacement data-name element (see "Processing for Address Reference (78) Elements" in this chapter). If the ADREF routine changes the element, the increment portion of the Incremented address (A4) element is written out as an Address increment (80) element with the high-order bit of the low-order byte set to 1 to indicate that the increment has already been added.

#### CONSTRUCTING DEBUG-TEXT

If the SYMDMP or the STATE option is in effect, Procedure A-text is used to create Debug-text which is written on SYSUT4. Debug-text elements are written by the SYS2 routine. For all Card number (2C) elements encountered, CARDLOC (10) elements are written that contain the card number and its displacement within the object module.



Debug-text also contains:

- Discontinuity (40) elements that are created when phase 63 combines two sections of equal priority that have discontinuous card numbers because of an intervening section or sections of different priority.
- ENDSEG (20) elements that identify the last byte of each segment, or the last byte of Procedure Division code if the program is not segmented.

Debug-text is used by phase 65 to produce debugging information for the COBOL library debugging subroutines.

### COUNTERS USED IN PHASE 63

While producing Procedure A1-text, phase 63 uses two counters: LOCCTR in COMMON and ACMCTR. LOCCTR is used to generate the relative displacements of each instruction listed and enter target addresses in the RLDTBL table. For details, see "Making Entries in the RLDTBL Table" in this chapter.

ACMCTR is incremented for all code that is to be contained in the completed machine language program. It is used to generate the displacements of PN and GN definitions within each separate Procedure block. Routines PNDEF and GNDEF build the PNLBDTBL and GNLBDTBL tables for this purpose. Phase 64 uses these tables to generate the proper displacements.

### BUILDING THE QGNTBL TABLE

The QGNTBL table lists the Q-Routine GNs and their corresponding Procedure block numbers. These are needed by phase 64 to initialize Q-Routines during INIT3 processing. The table is built from the GNLBTBL by phase 63 at each GN definition (34) element following the Q-BEGIN macro-type instruction (4440) element.

### MAKING ENTRIES IN THE RLDTBL TABLE

RLD entries are made to:

- Resolve VN addresses
- Resolve the GNs for REPORT-ORIGIN verbs

- Produce RLD-text for the linkage editor

An RLD entry contains the relative address within the object module for the entry item.

Before an entry is made, the RLDTBL table is sorted. RLDTBL entries are created by phases 63 and 64; the RLDTBL table is completed and processed by phase 64. Processing the RLDTBL table entails writing RLD-text in some cases. At object time, the linkage editor relocates addresses contained in the RLD-text. (Not all RLD entries cause RLD-text to be written.)

Routines PNDEF, GNDEF, and STARTMAC enter the relative address of an address constant in the RLDTBL table as well as the target address.

Routine D4 makes entries for locations associated with REPORT-ORIGIN verbs. It sets a bit to indicate to phase 64 that RLD cards for these entries are not to be produced. For details on these entries, see "Processing for RPT-ORIGIN (D4) Elements," which appeared earlier.

### PROCESSING IN A SEGMENTED PROGRAM

When a program is not segmented, phase 63 reads Procedure A-text from SYSUT1 in the order in which it was written. When a program is segmented, Procedure A-text is read in order of ascending priority so that the procedure instructions for the root segment are processed first.

The PHAS63 routine first determines whether the program is segmented by checking SEGLMT in COMMON. If SEGLMT does not contain X'FF', the routine relocates the SEGTLB table and indicates to phase 00 that a POINT macro instruction is to be issued to access the root segment in Procedure A-text. Processing of Procedure A-text begins at that point.

In phase 1B, the priority numbers of all sections in the root segment were set to 0.

When the MACRO routine comes to the end of a section, it branches to the SEGBRK routine, which determines whether the end of the section is also the end of a segment. If it is the end of the segment, the routine searches the SEGTLB table for the next segment of next highest priority. If it is not the end of the segment, the SEGTLB is searched for the next section of the same priority. When the end of the SEGTLB table is reached, control passes to routine EOF.

If the SYMDMP or the STATE option is in effect, at the end of processing for each segment, the final LOCCTR value for that segment and the priority for the next segment to be processed are both written on SYSUT4 for phase 65.

Note: A section is a series of source program procedure instructions grouped under the same section-name. A segment is all the instructions whose sections have the same priority, and a segment may consist of one or more sections. There is a SEGTBL entry for every section whose priority differs from that of the section preceding it.

#### PROCESSING AT END OF FILE

When all segments have been processed or at end of file in an unsegmented program, routine EOF calls routine RLDSORT to sort the final RLD entry.

Then it releases the tables used by phase 63, except for the PNLBDTBL, GNLDTBL, VNPTY, VIRPTR, LTLTBL, BLASGTBL, GNATBL, PNATBL, QGNTBL, and RLDTBL tables, which are passed to phase 64. Finally, it restores LOCCTR, and returns control to phase 00.

Phase 64 (IKFCBL64) is the last of the three phases that produce the machine language program. The major functions of phase 64 are:

- Completing the RLDTBL table and building the QTBL table by processing Data A-text.
- Processing E-text and DEF-text.
- Processing Procedure A1-text and entering displacements in the instructions generated by phase 63.
- Writing object text and REF-text from Procedure A1-text.
- Writing object text for the INIT2, INIT3, and INIT1 routines of the object program in that order.
- Writing object text and RLD-text from the RLDTBL table.

#### OUTPUT OF PHASE 64

An introductory discussion of the output generated by phase 64 in response to compiler options is given under "Output of Phases 62, 63, and 64" in the chapter "Phase 62."

#### PROCESSING DATA A-TEXT, E-TEXT, AND DEF-TEXT.

Phase 64 reads Data A-text from SYSUT4 before it reads Procedure A1-text from SYSUT2. It does this because Procedure A1-text for segmented programs has been written by phase 63 in order of ascending priority with the root segment first. Therefore, phase 64 must complete all RLDTBL entries for the root segment before Procedure A1-text is read.

The primary function of Data A-text processing is to place values into the data areas and fields of the Global Tables of

the object program. Each element results in either the writing of an object text element or an entry in the RLDTBL. Some RLDTBL entries will later be written out as Relocation Dictionary (RLD-text) entries for the data areas and as object text. Others, for the Global Tables, will be written as object text only. (These will be relocated by the object program.)

SYSUT4, from which Data A-text is read, also contains E-text generated by phases 10 through 51, and DEF-text for the cross-reference listing if the SXREF, XREF, VBREF, VBSUM, or TEST option is in effect. Figure 55 illustrates the contents of this data set when it is read by phase 64. Figure 56 illustrates how each type of element is processed.

#### PROCESSING PROCEDURE A1-TEXT

Phase 64 reads Procedure A1-text from SYSUT2 to produce the machine language instructions for the object program. One element of text is read and processed at a time, and the object code produced for this element is placed in a work area OU6REC. One or more elements are required to produce a complete instruction. When an instruction is complete, it is written out from the work area.

If the PMAP option is in effect, the PUT routine is called to write a line of text on SYSPRINT every time a complete instruction has been created. If the CLIST option is in effect, this routine is called only for each source program verb.

If the SXREF or the XREF option is in effect, Procedure A1-text is used to create REF-text, which is written on SYSUT3. This text, containing an element for every data-name, file-name, and procedure-name in the program, is used by phase 6A to produce a cross-reference listing.

Figure 57 describes the processing for each type of Procedure A1-text element. The individual elements are illustrated in "Section 5. Data Areas."



Code and Type	Action Taken
00 E-text	<p>All E-text is built into a table called ERRTBL, which is passed to phase 70. Phase 64 does not process the E-text. If the ERRTBL table overflows the space allotted to it, all of the E-text is rewritten on SYSUT3, and a bit in the SWITCH field of COMMON is set to indicate this to phase 70.</p> <p>There are two types of E-text elements: (1) message definitions and (2) message parameters. Message parameters are optional; however, if they occur, one or more message parameters immediately follow the message definition to which they apply (the uses of these elements are explained in the chapter "Phases 70, 71, and 72"). Phase 64 examines each element to determine its length, so that the correct number of bytes may be stored in the table. To do so, it checks the third byte of the element. If the byte contains a zero, the element is a message definition whose length is 8 bytes. If the third byte is nonzero, the element is a message parameter of variable length, and the length is determined from the value of the second byte (for the format of E-text and ERRTBL see "Section 5. Data Areas").</p>
04 DCB address	<p>Generate an RLDTBL entry that will cause the address of the DCB to be placed in the correct cell of the DCBADR field in the PGT at execution time. Get displacement of the DCBADR field from cell DCBCTR in COMMON (see "DCBADR Allocation" in the chapter "Phase 62") and use the DCB number to compute displacement of cell. Text element contains the value (relative address of the DCB) to be placed in the PGT cell.</p>
08 DECB address	<p>Generate an RLDTBL entry that will cause the address of the DECB to be placed in the correct cell of the DECBADR field of the TGT at execution time. Get displacement of the DECBADR field from cell DECBCTR in COMMON (see Figure 48 in the chapter "Phase 62") and use the DECB number to compute displacement of cell. Text element contains the value (relative address of the DECB) to be placed in the cell.</p>
0C block address	<p>Generate an RLDTBL entry that will cause the address of the buffer to be placed in the correct BL cell of the TGT at execution time. Get displacement of the BL field from cell BLCTR in COMMON (see Figure 48 in the chapter "Phase 62") and use the BL number to compute displacement of the cell. Text element contains the value (relative address of the buffer) to be placed in the TGT cell.</p> <p>If the value of the SIZE field of the element exceeds 1024 (SIZE specifies length of the block in fullwords), more than one BL has been assigned to the buffer. For each 1024-word area after the first, another RLDTBL entry is made. The second RLDTBL entry will cause the buffer address plus 4096 to be placed in the next BL cell of the TGT.</p>
14 FIB address	<p>Generate an RLDTBL entry that will cause the address of the File Information Block to be placed in the correct cell of the FIB field in the TGT at execution time. Get displacement of the FIB field from AMICTR cell in COMMON and use the FIB number to compute displacement of cell. Text element contains the value (relative address of the FIB) to be placed in the TGT cell.</p>

Figure 56. Processing Data A-text, E-text, and DEF-text (Part 1 of 2)

Code and Type	Action Taken
24 Working-Storage Section address	Generate an RLDTBL entry that will cause the address of the Working-Storage Section to be placed in the correct BL cell of the TGT at execution time. Get displacement of the BL field from BLCTR cell in COMMON (see Figure 48 in the chapter "Phase 62") and use the BL number to compute displacement of the item. Text element contains the value (relative address of the Working-Storage Section) to be placed in the TGT cell. If the value of the SIZE field exceeds 1024 (SIZE specifies the length of the Working-Storage Section in fullwords), more than one BL has been assigned. For each 1024-word area after the first, another RLDTBL entry is made. The second entry will cause the address plus 4096 to be placed in the next BL cell.
28 constant definition	Write object text that will place the value of the constant into a specified location in the data area at execution time. This type of element is used to fill some fields of DCBs and DECBS, and to initialize data items for which a VALUE clause was specified.
2C address constant definition	Generate an RLDTBL entry that will cause the address of the procedure-name (PN) or generated procedure-name (GN) to be placed in a specified location of the data area at execution time. This type of element is used to place pointers to routines in DCBs and exit lists.
34 Q-Routine identification	This type of element contains a GN number for Q-Routine. The elements are built into a table called QTBL. Each entry is resolved so that it contains the actual address of the routine rather than simply the GN number. This processing is identical to that for GN references in Procedure A-text (see Figure 57 in this chapter). When phase 64 generates the code of INIT3 (one of the execution-time initialization routines), it uses the QTBL and QGNTBL tables to generate a call to some Q-Routines to initialize the data and table areas affected by some OCCURS...DEPENDING ON data items, where the object of the DEPENDING ON option is an item in the Working-Storage Section.
38 BL reference	Generate an RLDTBL entry that will cause the displacement in the TGT of the BL number assigned to VSAM files to be placed in a specified location of the data area at execution time.
3C BLL reference	Generate an RLDTBL entry that will cause the displacement in the TGT of the BLL numbers assigned to VSAM files in the Linkage Section to be placed in a specified location of the data area at execution time. This type of element is used to complete the building of the FIB at execution time.
48 data-name or file-name DEF-text	This element is present only if the SXREF or the XREF option was specified. Each element is written out as it is encountered on SYSUT1, to be read by phase 6A. The chapter "Phase 6A describes how these elements are used.
4C procedure-name DEF-text	This element is present only if the SXREF or the XREF option was specified. Each element is written out as it is encountered on SYSUT1, to be read by phase 6A. The chapter "Phase 6A" describes how these elements are used.

Figure 56. Processing Data A-text, E-text, and DEF-text (Part 2 of 2)

Note on Base Registers for the PGT and TGT:  
At execution time, register 12 always points to the beginning of the PGT and register 13 always points to the beginning of the TGT. If the displacement of an item in the PGT or TGT exceeds 4096 bytes, an

OVERFLOW cell must be used. The OVERFLOW cells fields of both the PGT and TGT are at fixed displacements from registers 12 and 13, respectively. Which OVERFLOW cell is to be used is determined from the value of the displacement, for example, a value

from 4096 to 8191 bytes uses cell 1, from 8192 to 12,287 bytes uses cell 2, etc. An instruction is generated to load register 14 or register 15 from the OVERFLOW cell. Then, in the operand currently being processed, register 14 or register 15 is used as the base; and the displacement is decremented by 4096, 8192, etc.

contents and functions of these three routines are described in detail in "Appendix B: Object Module."

PROCESSING THE RLDTBL TABLE

After the initialization routines are generated (after Procedure Division or root segment processing), the RLDTBL table is processed. First, indirect address constants are resolved. Object text is written for items that are in the global tables. This text consists of address constant definitions that will be stored in the Global Tables at execution time. Because the addresses are relocated during program execution by routine INIT3, no RLD-text is required for these items with one exception. RLD-text is written for relocating all PROCEDURE BLOCK cells contained in the PGT. Object text is also written for data area address constants (obtained from address constant and indirect address constant definitions). For the data area address constants, RLD-text is written so that the linkage editor can relocate the addresses.

INITIALIZATION CODING GENERATION

After Procedure A1-text has been processed, initialization coding is generated by phase 64, using the GINIT1, GINIT2, and GINIT3 routines.

There are three initialization routines, called INIT1, INIT2, and INIT3. All three are resident in the root segment. INIT2 is generated first, followed by INIT3. In the generation of INIT3, the QTBL table is used to generate code that will call the Q-Routines at the beginning of program execution to initialize variable-length fields. INIT1 is generated last because it contains pointers to INIT2 and INIT3. The

Code and Type	Action Taken
2C <sup>1</sup> card number	Store in 3-byte cells OU6CDN and XPCDNO. If PMAP or CLIST are requested, read Listing A-text. Used to create an inline constant for TRACE instructions that call the COBOL library DISPLAY subroutine (ILBODSPO).
30 <sup>1</sup> PN definition	Using PN number as an index, look in PNATBL to get displacement in PGT of the cell for this PN. Create an RLDTBL entry that will place the current value of LOCCTR in the PGT cell. If there is no entry in the PNATBL, no RLDTBL entry is created.
34 <sup>1</sup> VN definition	Same as PN definition, using GN number and GNATBL.
38 <sup>1</sup> VN definition	Create an indirect RLDTBL entry from this element and the PN reference that follows it.
3C EBCDIC card name	Convert the current card number to an EBCDIC constant of the form: DC X'5' DC CL6'generated card number'
44 macro-type instruction	Use byte 2 of the element as index to a branch table. Phase 64 produces the required coding. The contents of these elements are listed in the Procedure A-text formats given in "Section 5. Data Areas."
<sup>1</sup> Indicates that no object text was written for this element.	
<sup>2</sup> See "Note on Base Registers for the PGT and TGT" in this section.	

Figure 57. Processing Procedure A1-text Elements (Part 1 of 4)

Code and Type	Action Taken
48 operation code	This element contains in machine language the first 2 bytes of an instruction. The first byte is the operation code; the second byte may give condition codes, registers, or other operands. For an RR type instruction, this element contains the complete instruction. It is written out as received.
4C PN reference	This is the operand of a LOAD instruction. Procedure branching is accomplished by loading an address and then branching to it. Using register12 <sup>2</sup> as a base, find displacement by using PN number as an index into PNATBL. Using card number stored in XPCDNO, write an element of REF-text for phase 6A if SXREF or XREF is in effect.
50 GN reference	Same as PN reference, using GN number and GNATBL. No REF-text is written.
54 VN reference	Use R13 <sup>2</sup> as base. Get displacement of VN field of TGT from VNLOC cell in COMMON (see "Task Global Table Storage Allocation" in the chapter "Phase 62"). Use a VN number to compute displacement of this VN cell.
58 virtual reference	Use virtual number as an index in the VIRPTR table (see "VIRTUAL Allocation" in the chapter "Phase 62"). Table entry contains displacement of this virtual in the PGT. Use register12 <sup>2</sup> as a base.
5C BL reference	This element is the operand of an instruction that loads a base register. Use register13 <sup>2</sup> as a base, get displacement of BLL or BL field in TGT from BLLCTR or BLCTR, respectively, in COMMON. Use BL number to compute displacement of this cell.
60 TGT standard area reference	Use register13 <sup>2</sup> as a base. Displacement is picked up from a list of constants. This element refers to a cell in the fixed portion of the TGT.
64 Global Table variable located area reference	Use register13 <sup>2</sup> as a base (unless the element specifies the DCBADR field of the PGT, which uses register 12 <sup>2</sup> ). Get displacement of the TGT or PGT field from the appropriate cell in COMMON, and use identifying number to compute displacement of this item (see "Task Global Table Storage Allocation" and Figure 48 in the chapter "Phase 62").
68 literal reference	Bytes 2 and 3 are used to find the correct entry in the LTLTBL table, which gives the displacement of this literal in the PGT. Register 12 is the base.
6C DC definition	This element is used to create an inline constant for a calling sequence. It is always preceded by the element 4424, the macro-type instruction element signaling a DC definition (see earlier in this figure). It is written out as received.
70 base and displacement	Specifies the actual register number and displacement for the instruction. It is written out as received.
<sup>1</sup> Indicates that no object text was written for this element. <sup>2</sup> See "Note on Base Registers for the PGT and TGT" in this section.	

Figure 57. Processing Procedure A1-text Elements (Part 2 of 4)



Code and Type	Action Taken
78 address reference	If the <i>i</i> field contains X'03', the BL, BLL, SBL, or SBS indicated is already loaded. Use the register indicated in the low-order 8 bits of the <i>d</i> field for the base register. Displacement is the <i>d</i> field of the element. If the <i>i</i> field contains a value other than X'03', save the contents of print buffers and generate load of register 14 or register 15 with the BL, BLL, SBL, or SBS indicated by the <i>k</i> field. If the high-order bit of the <i>i</i> field is on, use register 15; if it is off, use register 14. Displacement is the <i>d</i> field of the element. In either case, get the card number from XFCDNO to write an element of REF-text.
7C EBCDIC data-name reference	This element always follows the element 4404, the macro-type instruction element for ENTRY. It is used to punch an ESD-text type 1 card for the entry point.
80 address increment	If appended byte is not zero, print buffers are saved. One LA instruction is generated for each multiple of 4095 in the sum of the displacement saved for the 78 (or D0) and 80 elements. Buffer is restored; its displacement field is replaced by the amount in excess of the final multiple of 4095. Value of the 80 element is included in the symbolic field as "+NNN." If appended byte is not zero, no LA instruction is required since the displacement field is less than 4096.
84 relative address	This element is used to create an inline pointer to an item in a field of the TGT or PGT for a calling sequence. Get displacement of field from appropriate counter in COMMON and use identifying number to compute displacement of item.
A0 <sup>1</sup> register specification	Specifies the register used by a macro-type instruction element, and must follow certain of these elements (see the list of macro-type instructions under "Procedure A-text" "Section 5. Data Areas.")
A4 incremented address	This element combines the Address reference (78) and Address increment (80) elements into one (see those elements in this table).
B0 calling sequence displacement	Used to create an inline TGT or PGT pointer for a call to an object-time subroutine that requires a parameter containing a displacement from register 12 or register 13.
B4 <sup>1</sup> calling sequence dictionary pointer	Used when a file-name or data-name occurs in a calling sequence to write a REF-text element for phase 6A. Obtain card number from XFCDNO.
B8 <sup>1</sup> file reference	Used to write an element of REF-text.
BC segmentation and GO TO... DEPENDING ON call parameter	Generate 3 DC instructions, used as parameters by the GO TO...DEPEND- ING ON and Segmentation subroutines. Code generated is as follows: DC X'priority' DC X'Procedure block number' DC X2'displacement within Procedure block'
<sup>1</sup> Indicates that no object text was written for this element.	
<sup>2</sup> See "Note on Base Registers for the PGT and TGT" in this section.	

Figure 57. Processing Procedure A1-text Elements (Part 3 of 4)

Code and Type	Action Taken
C4 procedure block number	Add displacement within PGT of PROCEDURE BLOCK cell (or OVERFLOW cell) to each instruction that establishes addressability for a Procedure block. Use PRBLDISP cell in COMMON set by phase 62 for this purpose.
C8 procedure base register for PNs	Enter displacement in branch instructions generated by phase 63, using PNLBDTBL.
CC procedure base register for GNS	Enter displacement in branch instructions generated by phase 63, using GNLBDTBL.
D0 base displace- ment data-name	Specifies actual register number, displacement from start of area controlled by base register, and a data-name dictionary pointer. Write base and displacement, and branch to routine for processing dictionary pointer.
¹Indicates that no object text was written for this element.	
²See "Note on Base Registers for the PGT and TGT" in this section.	

Figure 57. Processing Procedure A1-text Elements (Part 4 of 4)

The function of phase 65 (IKFCBL65) is to produce debugging information which is used by object-time COBOL library debugging subroutines and by the IBM OS COBOL Interactive Debug Program Product (Program Number 5734-CB4). For information about the object-time COBOL library subroutines, see the publication IBM OS/VS COBOL Library Program Logic. The phase is given control only if the flow trace (FLOW), statement number (STATE), symbolic debug (SYMDMP), or interactive debug (TEST) compiler option is specified by the user. The transfer of control to phase 65 is described in "Processing Between Phases" in the chapter "Phase 00." The operations of phase 65 are described in Diagram 7 located with the foldouts at the back of this publication.

PROCESSING THE FLOW OPTION

If FLOW is specified, phase 65 obtains the number (n[n]) of traces requested from the FLOWSZ cell in COMMON. The number is stored in the first byte of the DEBUG TABLE in the TGT. The ILBOFLW0 subroutine requests storage for the Flow Trace table dynamically. For the FLOW option, the PNCHSW routine writes the flow trace information in the DEBUG TABLE. Further processing for the FLOW option is discussed in "Final Processing" later in this chapter.

COMMON PROCESSING FOR THE STATE, SYMDMP, AND TEST OPTIONS

When STATE is specified, the object-time statement number subroutine (ILBOSTN0) uses the PROCTAB and SEGINDX tables.

When SYMDMP is specified, the object-time symbolic debug subroutines use the Debug data set (SYSUT5) to produce the formatted symbolic dumps requested. When TEST is specified, the IBM OS COBOL Interactive Debug Program Product (Program Number 5734-CB4) uses the Debug data set. The Debug data set tables are created by phases 25 and 65 (see "Section 5. Data Areas" for a description of the Debug data set).

For the STATE option, phase 6 or 63 or for the SYMDMP or TEST option, phase 63 created Debug-text, which is used by phase

65 to produce tables that provide information needed by the STATE or SYMDMP COBOL library subroutines or the IBM OS COBOL Interactive Debug Program. (Phase 6 writes Debug-text on SYSUT2; phase 63 writes Debug-text on SYSUT4.) Phase 65 builds the PROCTAB and SEGINDX tables for either the STATE, SYMDMP, or TEST option. For the STATE option, the PROCTAB and SEGINDX tables are written in the object module; for the SYMDMP or TEST option, they are written on the Debug data set. The CARDINDX, PROCINDX, and PROGSUM tables are created only for the SYMDMP and TEST options and are written on the Debug data set. The OBODOTAB and DATATAB tables have already been created for the SYMDMP and TEST options by phase 25. For the TEST option phase 65 also creates the BCDPN table and writes it on the Debug data set.

PROCESSING DEBUG-TEXT

The RDF2 routine locates and reads the Debug-text, which is passed to phase 65 on SYSUT2 from phase 6 or on SYSUT4 from phase 63. Debug-text elements are described in "Section 5. Data Areas."

The F2PROCS branch table is used to branch to one of the routines that control the processing for the elements.

Control	Elements
<u>Routine</u>	<u>Processed</u>
TENPROC	CARDLOC
TWENPROC	ENDSEG
FRTYPROC	Discontinuity

Building the PROCTAB Table

The TENPROC routine builds the PROCTAB entries from the information in the CARDLOC elements. Each PROCTAB entry contains the relative address of the instruction generated for the card and verb number in the entry. Phase 65 divides any program or segment that exceeds 64K bytes in size into fragments less than 64K bytes in length and creates a final (dummy) PROCTAB entry to indicate the end of the each fragment.

### Building the SEGINDX Table

A SEGINDX entry is created for each fragment of the program.

If the TENPROC routine determines that the code generated for the last verb causes the current fragment to exceed the maximum size (64K bytes), it calls the GTEQ10K routine to handle the processing for the end of the fragment. The GTEQ10K routine calls the SNF routine to start the new fragment, make a SEGINDX entry for the old fragment, and begin collecting information for the next SEGINDX entry.

The end of the Procedure Division is signaled by an ENDSEG element. When the RDF2 routine reads an ENDSEG element, it calls the TWENPROC routine to process the ENDSEG.

### FURTHER PROCESSING FOR THE STATE OPTION

If STATE is specified, the PROCTAB and SEGINDX tables are created and written in the object module as described above. Addresses passed in the TGTADTBL table are used to write the PROCTAB and SEGINDX tables in the object module. The TXPNCH routine writes the PROCTAB table in the object module following INIT3. At end of file for Debug-text, the EOF2 routine writes the SEGINDX table, after the PROCTAB table in the object module. The addresses of the beginning of each of the PROCTAB and SEGINDX tables and of the end of the SEGINDX table are saved in the DEBUG TABLE in the TGT.

The discussion of processing for the STATE option continues in "Final Processing" later in this chapter.

### FURTHER PROCESSING FOR THE SYMDMP OR TEST OPTIONS

If SYMDMP or TEST is specified, phase 65 processes the CARDINDX, PROCINDX, and PROGSUM tables for the Debug data set. Processing for the CARDINDX and PROCINDX tables occurs in conjunction with the processing for the PROCTAB and SEGINDX tables. If TEST is specified, phase 65 then builds the BCDPN table. The PROGSUM table is processed after the other tables have been written on the Debug data set.

### Building the CARDINDX Table

The CARDINDX table contains an entry for each fragment of the program and for each discontinuity in the COBOL instructions within a segment of the program.

The discontinuity elements in Debug-text indicate the discontinuity in card numbers at the end of each noncontiguous section.

When the RDF2 routine reads a discontinuity element, it branches to the FRTPROC routine. This routine sets the DISCSW switch to indicate this special processing is to be done for the end of section.

CARDINDX entries are created for discontinuity within segments and for each program fragment.

### Building the PROCINDX Table

After the TXPNCH routine moves a PROCTAB element into the SYSUT5 buffer, it determines whether the buffer is full. If the buffer is full, it calls phase 00 to write the buffer and builds a PROCINDX entry providing card and verb number information about the first entry in the block and the note address of the block after it has been written. The NOTE address of the first block of the PROCTAB table is saved for the PROGSUM table in the PROCTNTE save area.

### DEBUG DATA SET PROCESSING

The TXPNCH routine writes the PROCTAB table on the Debug data set at the beginning of a new block.

At end of file on SYSUT4, control is transferred to the EOFON2 routine to collect information about the number of entries in each of the CARDINDX, SEGINDX, and PROCINDX tables. It stores this information for the PROGSUM table in the CARDINUM, SEGINUM, and PROCNUM save areas, respectively. It then sorts the CARDINDX table in order of ascending card number priority.

The EOFON2 routine then moves the CARDINDX, SEGINDX, and PROCINDX tables (in that order) to the buffer for the Debug data set (SYSUT5). (These tables are written on the Debug data set, beginning at a new block.) The EOFON2 routine saves the displacement within the buffer of the

startof the SEGINDX and PROCINDX table in the SEGDSPL and PROCDSPL save areas, respectively. This information becomes part of the PROGSUM table.

The EOFON2 routine then calls phase 00 to write the tables and note those blocks that contain the beginning of a table. It saves the note information in the CARDNOTE, SEGNOTE, and PROCNOTE save areas for the PROGSUM table.

All the information gathered by the EOFON routine is entered in the PROGSUM table.

If the TEST option is specified, phase 65 then reads DEF-text from SYSUT1 and builds the BCDPN table. The table is written on the Debug data set (SYSUT5). The device address of the first block of the table, along with the total number of blocks in the table, is inserted into the PROGSUM table.

If the Debug data set is located on disk, the first 512-byte record is read back into the buffer, and the PROGSUM table is moved into the first 108-byte field. The record is then rewritten on the disk. This processing is done in phase 65.

If the Debug data set is located on tape, the ENDOFTBL routine issues a request to phase 00 to write the end of file mark and reposition both SYSUT5 and SYSUT2 to the first record. Phase 00 is requested to read the first 512-byte record of SYSUT5

into the buffer and phase 65 inserts the PROGSUM table in the first 108 bytes. It calls phase 00 to write the buffer on SYSUT2 and to copy the remainder of the contents of SYSUT5 onto SYSUT2. Then it recopies SYSUT2 onto SYSUT5.

#### FINAL PROCESSING

For any of the options, the PNCHSW routine sets the fullword SWITCH in the TGT to reflect the options in effect and it also sets the DEBUG TABLE PTR in the TGT. If an error has occurred, the DEBUG TABLE PTR is set to zero and the corresponding bit in the PH6ERR cell in COMMON is set on. (Phase 70 examines the bits in PH6ERR to determine which error messages, reflecting errors detected in phase 6, 62, 63, 64, or 65, are to be written.) The DEBUG TABLE is created from information produced during phase 65 processing and from information in the TGTADTBL table, and it is written in the TGT. Finally, the END card for the program is written in the object module, and if both the BATCH and NAME options have been specified, a linkage editor control card is written for the object module. If the program is segmented, the object code for independent segments is obtained from SYSUT1 and written in the object module. If TEST is in effect, phase 65 inserts the DEBUG TABLE information in the TGT of the object module.

## PHASE 6A

The function of phase 6A (IKFCBL6A) is to produce a cross-reference listing on SYSPRINT. The phase is given control only if the SXREF, XREF, VBREF, or VBSUM compiler option was specified by the user. The transfer of control is described under "Processing Between Phases" in the chapter "Phase 00."

Phase 6A tests the PHZSW1 byte in COMMON to determine whether the option specified is SXREF or XREF. If SXREF is in effect, phase 6A generates an alphabetically ordered cross-reference listing. Phase 6A tests the PHZSW4 byte in COMMON to determine whether the option specified is VBREF, or VBSUM. If VBREF is in effect, phase 6A produces the verb cross-reference listing. If VBSUM is in effect, phase 6A produces the verb summary listing. If XREF is in effect, a cross-reference listing ordered by source statement sequence is generated.

Phase 6A performs the following operations:

- Reads DEF-text into storage until either storage is filled or end-of-file is reached.
- Creates a DATA record and, if SXREF is in effect, a CONTROL record, for each DEF-text element. For SXREF, the CONTROL records are used in sorting the external names.
- Reads REF-text and appends references to the proper DATA record (or OVERFLOW record) if the definition has been read into storage.
- Prints each DATA record and its associated OVERFLOW records.

These operations constitute the fundamental cycle that is repeated until all DEF-text has been processed.

DEF-text for verbs has the same format and is processed in the same manner as DEF-text for data-names. Similarly, REF-text for verbs (VBREF only) has the same format and is processed in the same manner as REF-text for data-names. The only special processing for verbs is the recognition of the first verb-element and the first procedure-name thereafter.

During phase initialization, phase 6A uses the GETALL routine in TAMER (see the

chapter "Table and Dictionary Handling") to get all available storage for use in creating the DATATBL table, OFLOTBL table, and the CNTLTBL table (if the SXREF option is in effect), which contain the DATA records, OVERFLOW records, and CONTROL records, respectively. However, it does not use TAMER routines to access these tables. Therefore, phase 6A is able to construct tables occupying more than 32K bytes.

### PRODUCING A SOURCE ORDERED CROSS-REFERENCE LISTING

The maximum amount of space is obtained for the DATATBL and OFLOTBL tables by a call to GETALL in phase 00.

The DEF-text is read from SYSUT1, where it was written by phase 6 or 64. There is one element of DEF-text for each data-name, file-name, and procedure-name in the source program. The text on SYSUT1 is read into storage until either storage is filled or end-of-file is reached. One DATA record is created for each DEF-text element.

The REF-text is read from SYSUT3 or, if OPT is in effect, from file SYSUT1. There is one element of REF-text for each time the name is referred to in the source program. The text on SYSUT3 is read, one element at a time, until end-of-file is reached.

For a data-name or file-name, the internal name is the dictionary pointer assigned by phase 22. For a procedure-name, the internal name is the PN number assigned by phase 1B. The setting of a bit in each entry indicates whether it contains a dictionary pointer or a PN number. When a REF-text element is read, the high-order bit of the referencing card number is tested to determine whether the reference is to a data-name or to a procedure-name. (This test is made in case a dictionary pointer and a PN number were assigned the same value.)

If the DEF-text element for the referenced data-name or procedure-name was processed in this cycle, the DATA record is in storage. For a REF-text element for a data-name, a binary search of the DATATBL table is made to locate the matching DATA record for the data-name. A REF-text

element for a procedure-name is matched directly by means of an algorithm with the DATA record for the procedure-name. If a match is not found, the REF-text element is ignored. If a match is found, the referencing card number contained in the REF-text element is placed in the DATA record, or if it is full, in an OVERFLOW record chained to it. If the current OVERFLOW record is full, another OVERFLOW record is added to the chain, and the referencing card number is inserted in the first three bytes of the record. If no OVERFLOW record is available, the last DATA record is split into three OVERFLOW records which are then placed on the overflow-free chain. Before the referencing card number is placed in the newly designated OVERFLOW record, the REF-text element is rechecked to ensure that the matching DATA record was not just deleted. At end-of-file, the REF-text data set is closed.

At the end of the cycle, each DATA record and its associated OVERFLOW records are printed on SYSPRINT.

If this is the last or only cycle, processing is completed when all names, defining card numbers, and referencing card numbers in storage have been printed. If this is not the last cycle, DEF-text is again read into storage. (If it was necessary to split one or more DATA records into OVERFLOW records in the preceding cycle, the DEF-text data set must be closed, rewound, and reopened.) Names are read and ignored until the last name processed in the preceding cycle is reached. DATA records are created for unprocessed names and the cycle continues with the reading of REF-text.

#### PRODUCING AN ALPHABETICALLY ORDERED CROSS-REFERENCE LISTING

The maximum amount of space is obtained for the DATATBL, OFLOTBL, and CNTLTBL tables by a call to GETALL in phase 00.

The DEF-text is read from SYSUT1. The text is read into storage until either storage is filled or end-of-file is

reached. One DATA record and one CONTROL record are created for each DEF-text element. The CONTROL records are used in sorting the external names, which is done as the DEF-text is read.

The REF-text is read from SYSUT3. The text on SYSUT3 is read, one element at a time, until end-of-file is reached and the bit is tested as described in "Producing a Source Ordered Cross-Reference Listing." A search of the DATATBL table is made for the matching DATA record and if it is found, the referencing card number is placed in the DATA record, or if it is full, in an OVERFLOW record chained to it. If the current OVERFLOW record is full, another OVERFLOW record is added to the chain, and the referencing card number is inserted in the first three bytes of the record. If no OVERFLOW record is available, the DATA record that is last on the chain of sorted DATA records is split into three OVERFLOW records which are placed on the overflow-free chain. Before the referencing card number is placed in the newly designated OVERFLOW record, the REF-text element is rechecked to ensure that the matching DATA record was not just deleted. If a matching DATA record is not found, the REF-text element is ignored. At end-of-file, the REF-text data set is closed.

At the end of the cycle, each DATA record and its associated OVERFLOW records are printed in alphabetic order on SYSPRINT. The lines printed give the external name (from the DEF-text element), the card number of the statement in which the item was defined (from the DEF-text element), and the card numbers of all the references (from the REF-text elements).

If this is not the last or only cycle, DEF-text is again read into storage. (Whenever more than one cycle is required to process the entire DEF-text data set, the data set must be closed, rewound, and reopened.) Names are read and each name is compared with the last name processed in the preceding cycle. If the name read alphabetically precedes the last name processed, it has already been processed. If the name read alphabetically follows the last name processed, the cycle continues with the creation of DATA and CONTROL records.

PHASES 70, 71, AND 72

Phases 70, 71, and 72 (IKFCBL70, IKFCBL71, and IKFCBL72) combine to generate all the compiler diagnostic messages for source program errors. Phase 70 contains all the codes except for the message texts for error messages generated by phases 20 through 65 and the associated address constant tables used to locate the text for any one of these messages. Phase 70 resides in storage during the entire message generation process and loads either phase 71 or 72 when message text is needed from one of these phases. Phase 71 contains the message texts for errors arising during processing by phase 20, 22, 21, or 25; phase 72 contains the message texts for phases 3, 35, 4, 45, 50, 51, 6, 62, 63, 64, or 65. Input to phase 70 consists of E-text, passed from phases 04 through 6A, which is either in storage or on SYSUT3 or SYSUT4. Output consists of completed messages, which are written on SYSPRINT and/or SYSTEM. Phase 70 also produces a listing of all error messages in numbered order if it finds that the name of the program is ERRMSG.

If the compiler options are such that phase 6 or phases 62, 63, and 64 produce no output, then phase 70 reads the E-text from SYSUT4. Otherwise, phase 6 or 64 reads in the E-text on SYSUT4 along with Data A-text and DEF-text. To avoid an extra input/output operation, phase 6 or 64 attempts to save the E-text in storage for phase 70 in the ERRTBL table. The ERRTBL table, however, has a maximum size; if all the E-text cannot be saved in it, phase 6 or 64 puts the E-text for phase 70 on SYSUT3.

If errors are encountered by phases 6, 62, 63, 64, or 65, the corresponding bits in the PH6ERR cell in COMMON are set to 1. No E-text is written. Instead, phase 70 tests the bits in PH6ERR to determine which messages are to be written and then generates these messages in the usual manner. Note that since there are only 10 bits, phase 70 can only produce ten messages for phase 6. Any others must be produced by phase 6 itself. The presence of the text in phase 72 for these additional messages is for the error message listing only.

INPUT FROM PRIOR PHASES

Phases 04 through 51 produce E-text in the same manner. Whenever a processing routine detects a source program error, it writes out E-text message definition elements. If parameters are associated with the error message, the phase writes E-text message parameter elements directly after the message definition (see "Section 5. Data Areas" for the formats).

Phase 04 writes E-text for BASIS and COPY statements on SYSUT3. Phase 10 writes E-text interspersed with Data IC-text on SYSUT3. When phase 21 reads the Data IC-text, it writes the E-text back onto SYSUT4 without change, along with Data A-text and its own E-text. Phase 1B writes the E-text produced during P0-text processing on SYSUT2 with the P0-text. From then on until phase 51, E-text is added to the Procedure IC-text stream as errors are encountered. Phase 51 separates E-text and writes it, as well as its own E-text, on SYSUT4. Note, however, that the E-text is not interspersed with other texts if the CSYNTAX or SYNTAX option is in effect, instead phases 4, and 50 write E-text directly on SYSUT4.

PHASE 70 ERROR PROCESSING

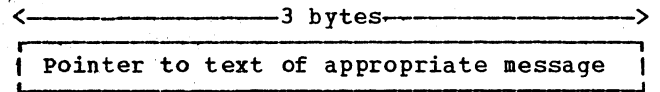
Upon receiving control from phase 00, phase 70 uses the PARTBL, EACTBL, and PHxERR tables together with E-text to construct error messages for the listing. The PH1ERR table is located in phase 70; the PH2ERR table is located in phase 71; and the PH3ERR, PH4ERR, PH5ERR, and PH6ERR tables are located in phase 72. Messages for error conditions that arise during processing by phase 01, 04, 10, 12, or 1B are completely processed within phase 70.

When an E-text element produced during phase 20, 22, 21, or 25 processing is encountered, phase 70 loads phase 71 into storage and picks up the address of the PH2ERR table which is located at the entry point IKFCBL71. Phase 70 stores these addresses in the corresponding entry in the PHMESS table and sets a switch to indicate that phase 71 has been loaded. Phase 70 can then use the pointers in the PH2ERR table to find the message texts contained in phase 71.



When an E-text element produced during phase 3, 35, 4, 45, 50, 51, 6, 62, 63, 64, or 65 processing is encountered, phase 70 loads phase 72, to obtain the addresses of the PH3ERR, PH4ERR, PH5ERR, and PH6ERR tables as for phase 71. These addresses are entered in the PHMESS table and the switch is set to indicate that phase 72 has been loaded. Phase 70 can then use the pointers in the corresponding PHxERR table to find the message texts contained in phase 72.

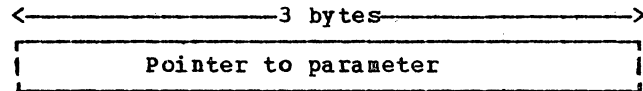
Each entry is of the following form:



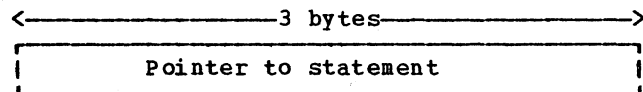
The message text is found in phases 70, 71, and 72 (as described earlier in this chapter).

PARTBL and EACTBL Tables

The PARTBL table is a fixed table assembled as part of phase 70 and is not handled by TAMER routines. It contains 3-byte pointers to all possible error message parameters, COBOL words, verbs, operations, etc., that are not programmer-supplied names. Its entries are of the form:



The EACTBL table is a fixed table assembled as part of phase 70 and not handled by the TAMER routines. It contains 3-byte pointers to error statements that describe which compiler action was taken because of the error, for example, "STATEMENT ACCEPTED AS WRITTEN." Its entries are of the following form:



PHxERR Tables

The PHxERR tables are fixed tables assembled as part of phase 70, 71, and 72 and not handled by TAMER routines. For each message, an entry is made in the appropriate PHxERR table, where x has the following values:

Value of x	Phase
1	04,10,12, or 1B
2	20,22,21, or 25
3	3 or 35
4	4 or 45
5	50 or 51
6	6,62,63,64, or 65

GENERATING MESSAGES

The XNORML routine scans each E-text item in turn. It first moves the card number, message-phase number, and the severity code to the work area XU6REC. The severity code is of the form:

IKFPxxxI - \*

where:

p is the phase number in which the error occurred, where:

Value of p	Phase
1	04,10,12, or 1B
2	20,22,21, or 25
3	3 or 35
4	4 or 45
5	50 or 51
6	6,62,63,64, or 65

xxx is the number of the message

\* is the severity code, as follows:

- W = warning
- C = conditional
- E = error
- D = disaster

The XNORML routine next uses the pointer in the corresponding PHxERR table entry for the E-text entry to find the text of the message itself. It places the text in work area XU6REC. It scans the text entry for the presence of the symbols \$, =, and /. The symbol \$ indicates that a parameter following the E-text for this message must be inserted at this point. Parameters are taken either from the parameter entry itself or from the location pointed to indirectly through the PARTBL table by the value field of the parameter entry.

The symbol = indicates that an error action message must be added. In this situation, a number directly follows the = symbol. Phase 70 uses this number as a pointer to determine the displacement into the EACTBL table for the pointer to the appropriate error action message.

The symbol / indicates that this is the end of the text for this message.

The error action messages are in phase 70 starting at location EACT00. Routine XNORML moves the error action message into the work area immediately following the text for the message. The routine XPUT then writes the message on SYSPRINT and/or SYSTEM.<sup>1</sup>

-----  
<sup>1</sup>Routine XPUT calls phase 00 with a request to write to SYSPRINT. Phase 00 then determines whether SYSPRINT, SYSTEM, or both are to be written to and takes the appropriate action.

#### ERROR MESSAGE LISTING

On entry to phase 70 it determines if the program-id stored in COMMON is ERRMSG. If so, it is taken as a sign that this is a dummy compilation simply to produce a listing of all error messages in numerical order. This is done by scanning PHXERR and SEVTBL sequentially.

#### FIPS PROCESSING

The FIPS processing phases (8s) cannot handle programs with syntax errors in the source. If phase 70 finds any errors of greater severity than W, a flag is set in COMMON (INFOHSG) to prevent FIPS processing. In addition some warning messages cause the flag to be turned on. Those that do not, have a bit (X'20') set on in their SEVTBL entry, indicating an informational message.

The function of phase 80 is to flag the COBOL source statements that are at variance with the Federal Information Processing Standard (FIPS). The phase is given control only if the LVL compiler option is specified by the user or was defined as the default value at installation time.

Phase 80 tests the FIPLVL switch in COMMON to determine which level of flagging has been specified (A = low; B = low intermediate; C = high intermediate; D = full standard).

Input

Input to phase 80 is the COBOL source program listing on the SYSUT6 utility data set and the FIPS parameter list. If the LSTCOMP option is in effect, the beginning of the Lister output (from phase 08) is identified by four bytes of binary zeros; the end of the Lister output is identified by four bytes of binary ones.

Phase 02 changes the name of the output data set for the source program listing from SYSPRINT or SYSTEM to SYSUT6 or their alternate DDnames when FIPS processing has been requested. The source program is written by phases 10, 12, and 1B.

The FIPS parameter list is described below:

Byte	Contents
0	Address of COMMON
4	Address of DDname for input to phase 80 (SYSUT6)
8	Address of DDname for output from phase 80 (SYSPRINT or SYSTEM)
12	Address of DDname for phase 80 work data set (SYSUT1)

The end of input to phase 80 is identified by the appearance of 999999 as the source sequence number.

Output

The output of phase 80 consists of the COBOL source program listing flagged according to the specified level of the Federal Information Processing Standard on SYSPRINT or SYSTEM.

Processing

If INFOMSG in COMMON is on, IKFCBL80 puts out error message number IKF8008 does not produce the source program.

Phase 80 performs the following functions:

- Scans each Division of the COBOL source program.
- Generates diagnostic messages for exceptions to the standard.
- Writes the source program and messages on SYSPRINT or SYSTEM.
- Performs input and output operations.
- Returns control to phase 00.

SCANNING THE SOURCE PROGRAM

The source program is scanned by the four scanning routines of phase 80. They are IDSCAN, ENVSCAN, DATASCAN, and PROCSCAN; they process the Identification, Environment, Data, and Procedure Divisions, respectively. These routines are under control of the IKFCBL80 routine and they use the GETLINE, PUTLINE, GETWORD, CHKCOPY, CHKGLBLS, MSGHNDLR, and VERBCHK routines.

The scanning routines call the GETLINE routine to read a line of the source program and the GETWORD routine to determine each word of the line. Subroutines in each scanning routine check each word to see if it meets the FIPS standard. Diagnostic messages are issued for exceptions to the standard.

GENERATING DIAGNOSTIC MESSAGES

When an exception to the FIPS standard is discovered in the source program, the MSGHNDLR and MSGWRITE routines are called to format the diagnostic message and to write it on the output data set.

Each Division scanning routine contains the text of the messages that are issued by that routine. When the MSGHNDLR routine is called, the address of the message is

passed to the routine and the routine formats it for printing by the MSGWRITE routine.

#### WRITING THE SOURCE PROGRAM

Each time the GETLINE routine is called by one of the scanning routines, it reads a

line of the source program and then calls the PUTLINE routine to write the line on SYSPRINT or SYSTEM. SYSTEM is used only if the user has specified the TERM option, or when operating under TSO or CMS. After the line is written on the output data set, control is returned to the scanning routine for FIPS processing.

SECTION 3. PROGRAM ORGANIZATION

FLOWCHARTS

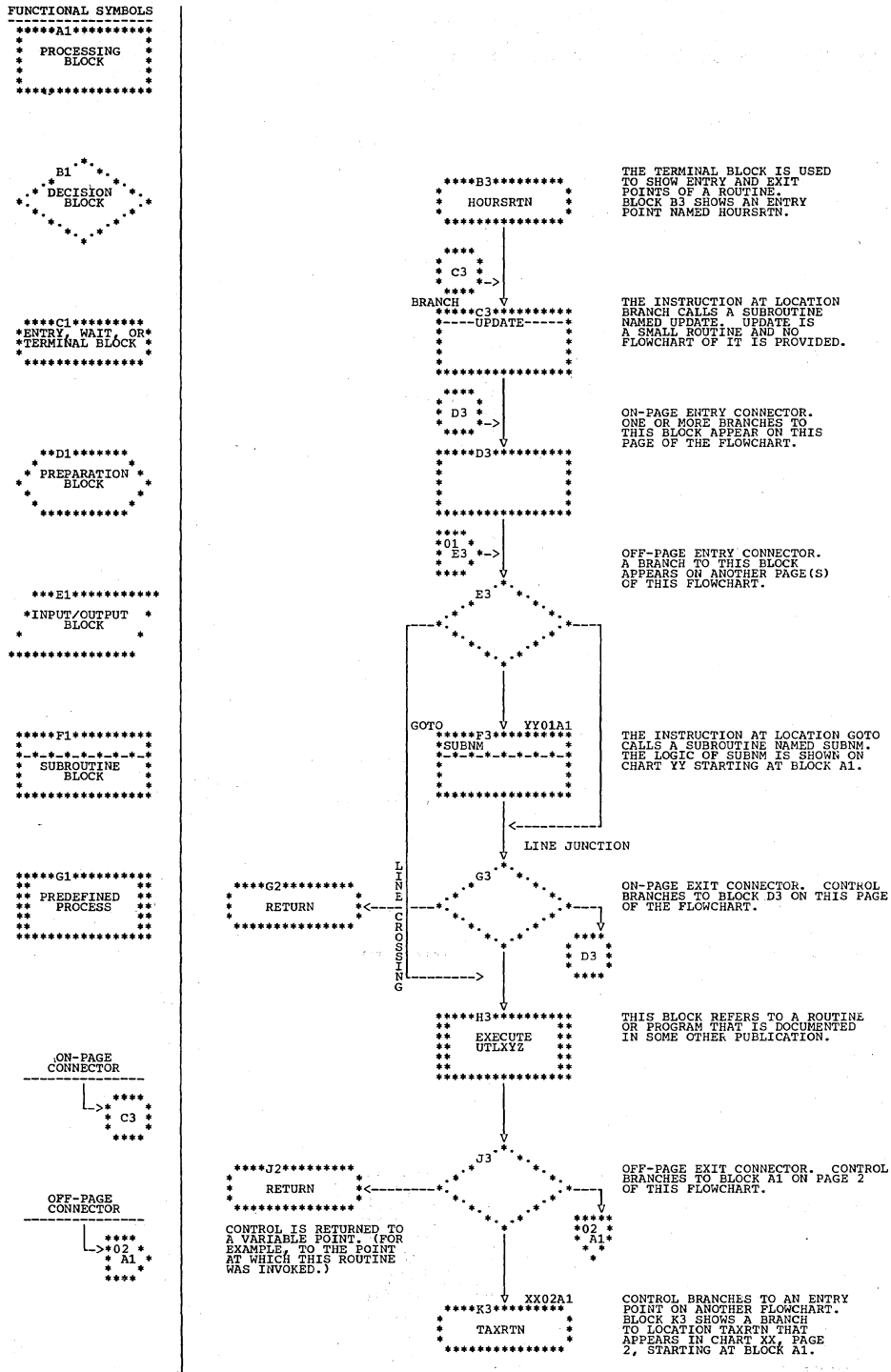


Figure 58. Explanation of Flowchart Symbols

Chart AA (Part 1 of 7). Phase 00: Overall Flow

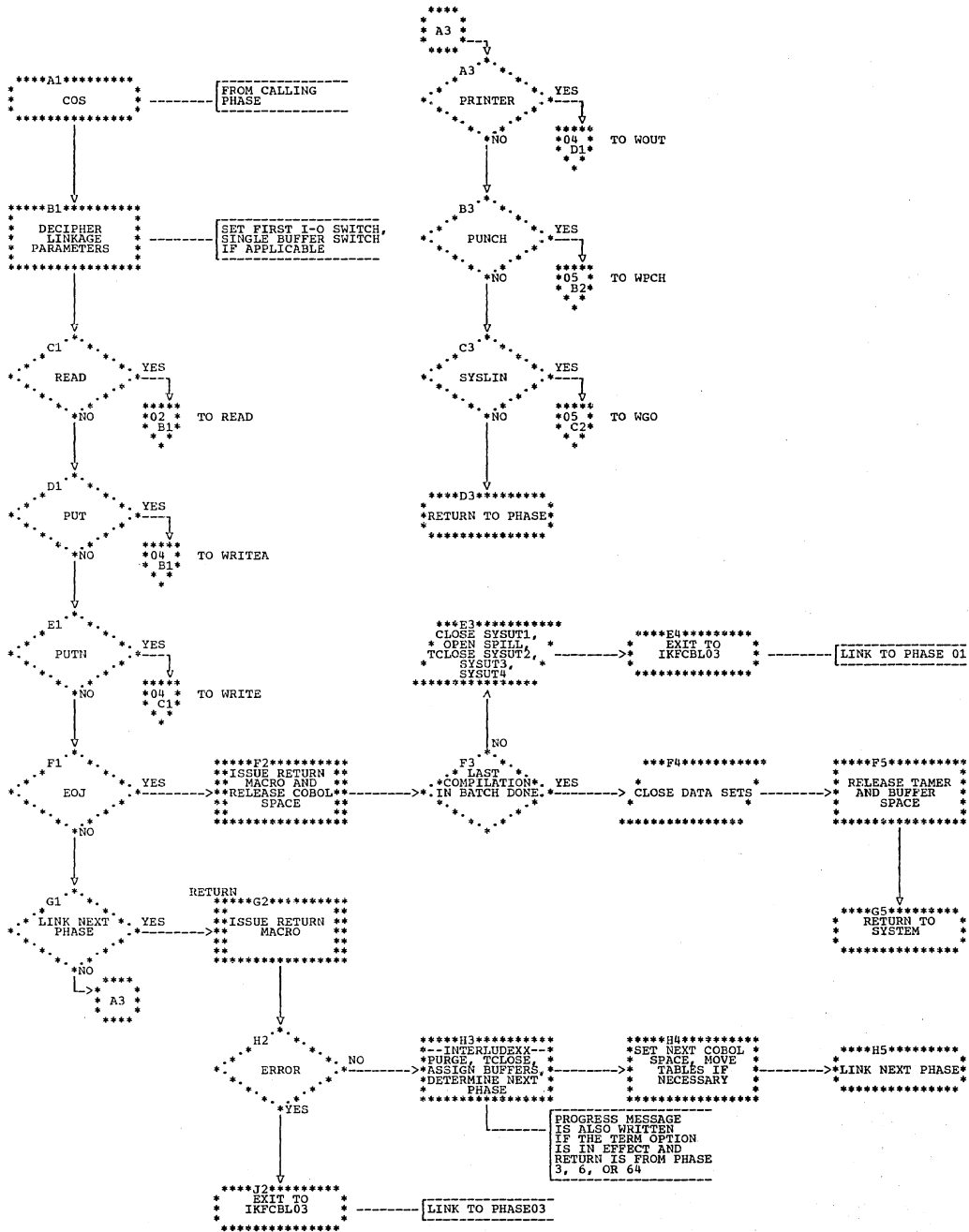


Chart AA (Part 2 of 7). Phase 00: READ Routine

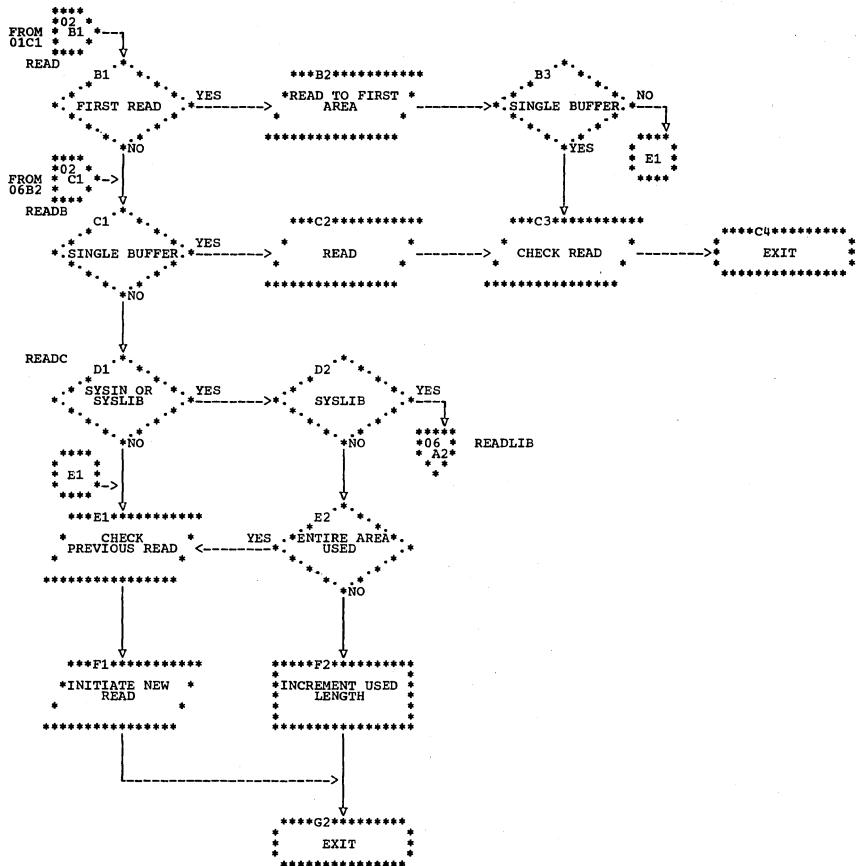




Chart AA (Part 3 of 7). Phase 00: WRITEA, WRITE, and WOUT Routines

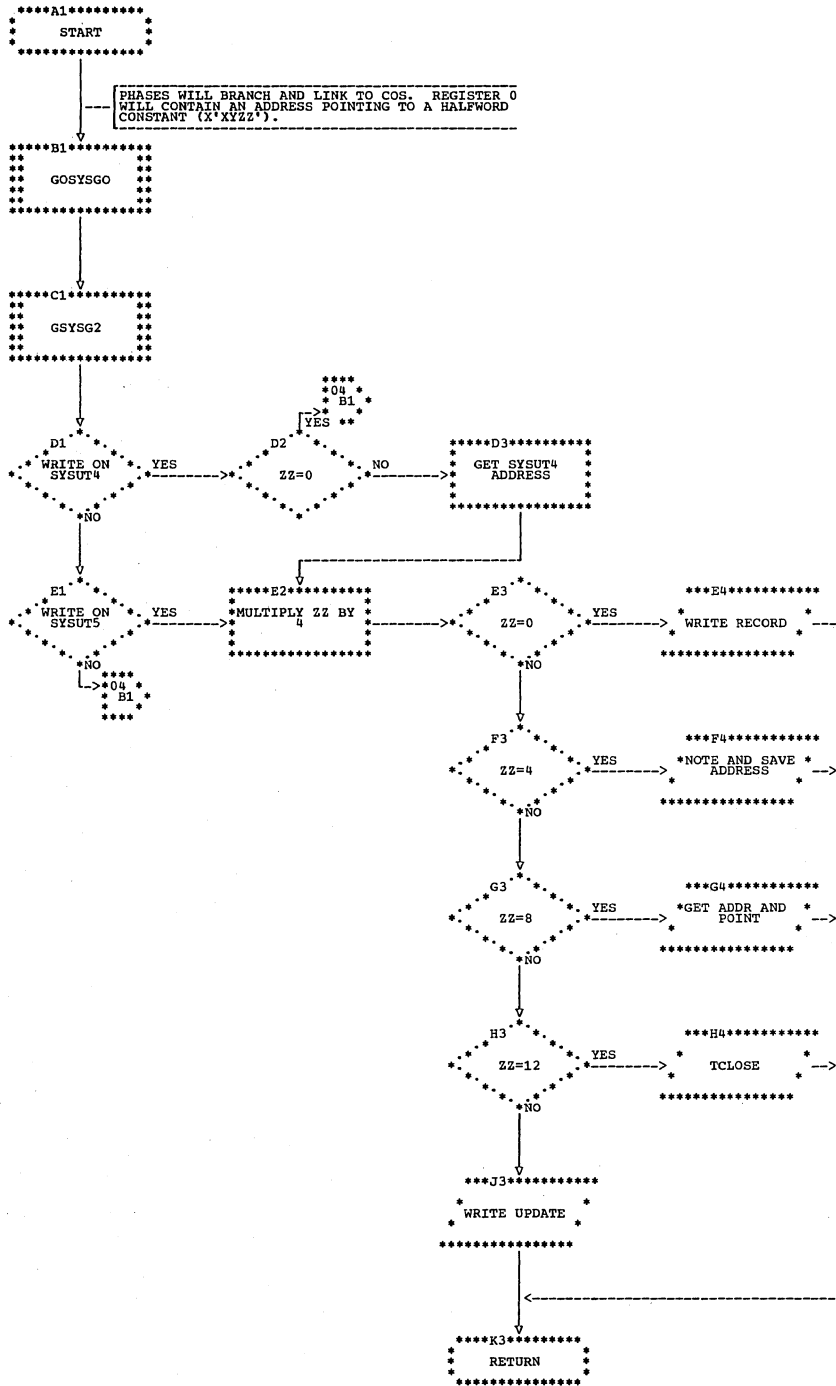


Chart AA (Part 4 of 7). Phase 00: WRITEA, WRITE, and WOUT Routines

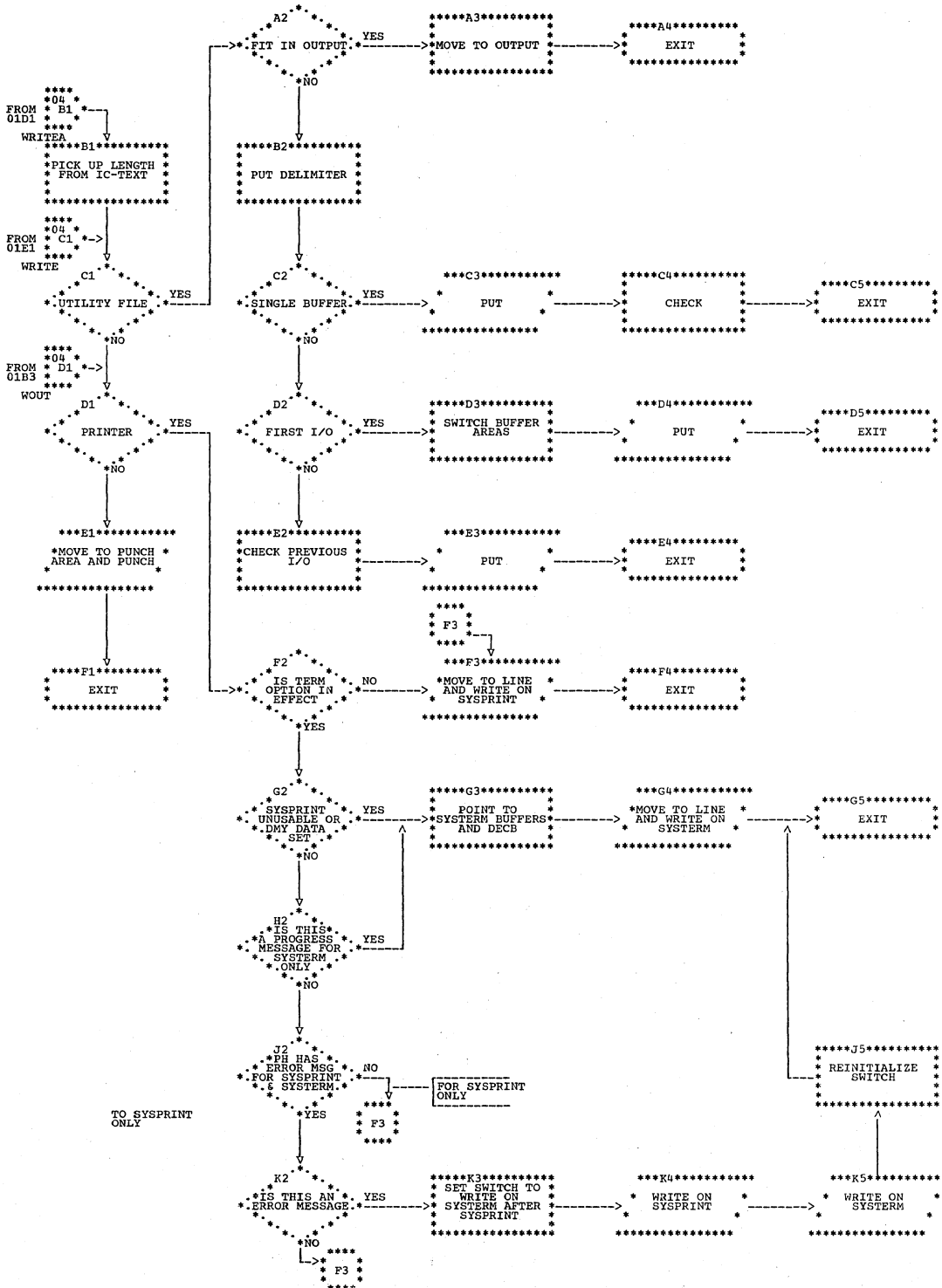


Chart AA (Part 5 of 7). Phase 00: WPCH and WGO Routines

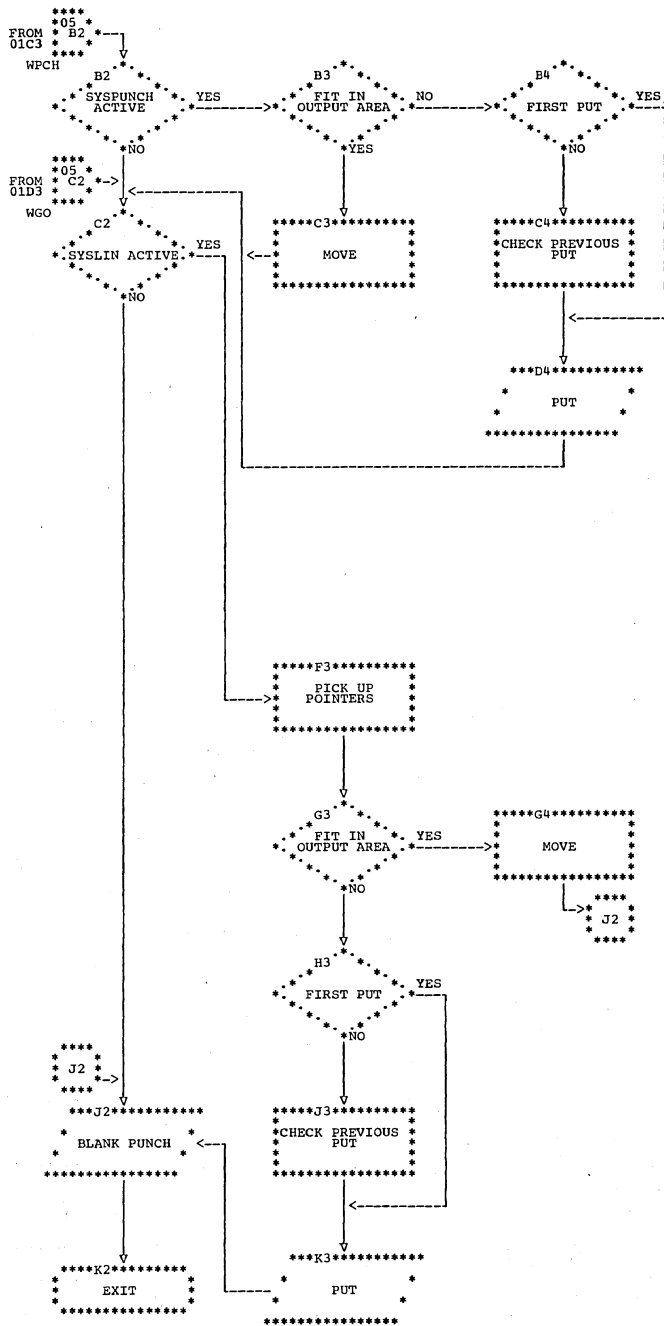
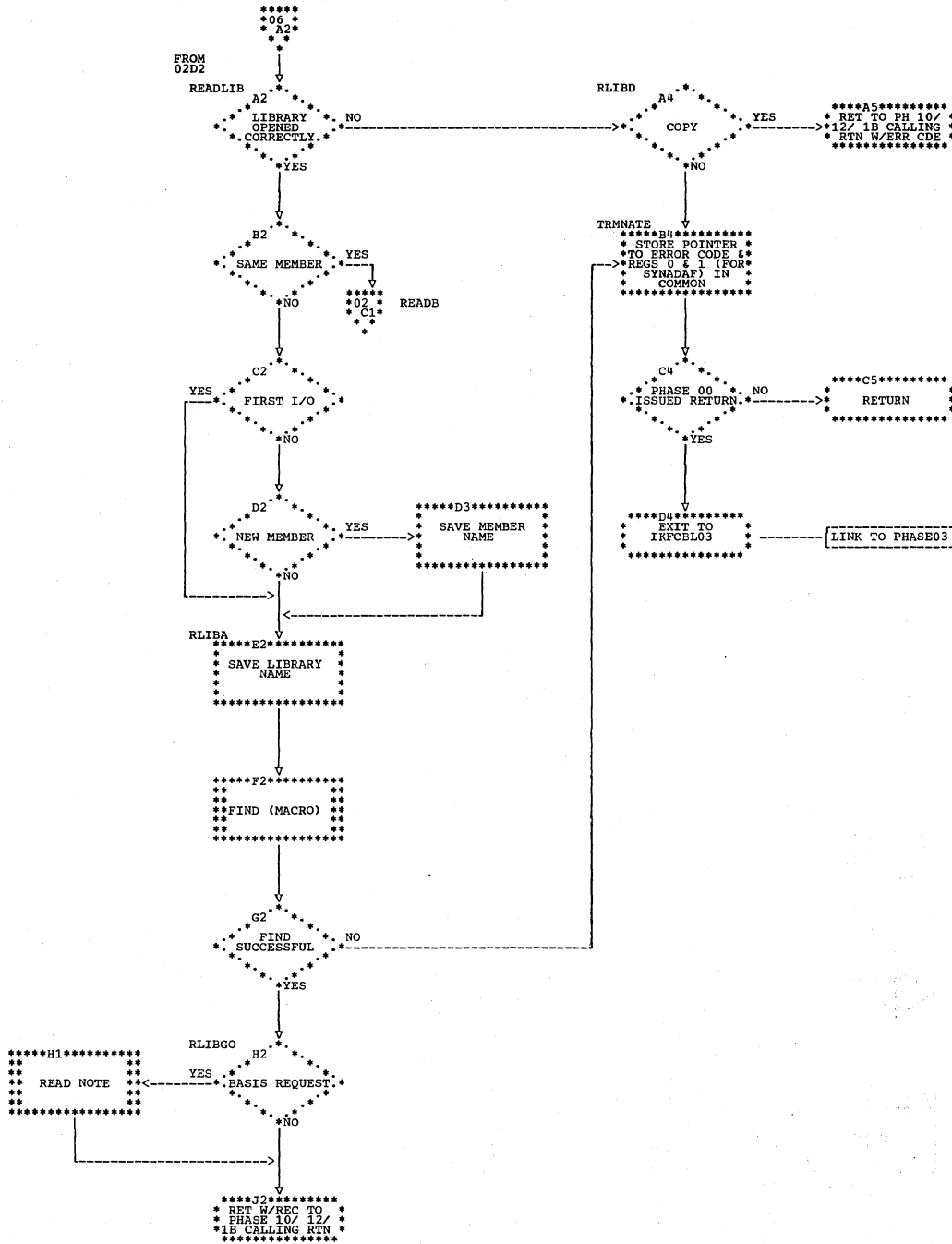


Chart AA (Part 6 of 7). Phase 00: READ Library Routines



| Chart AA (Part 7 of 7). Phase 00: PLSCALL Routines

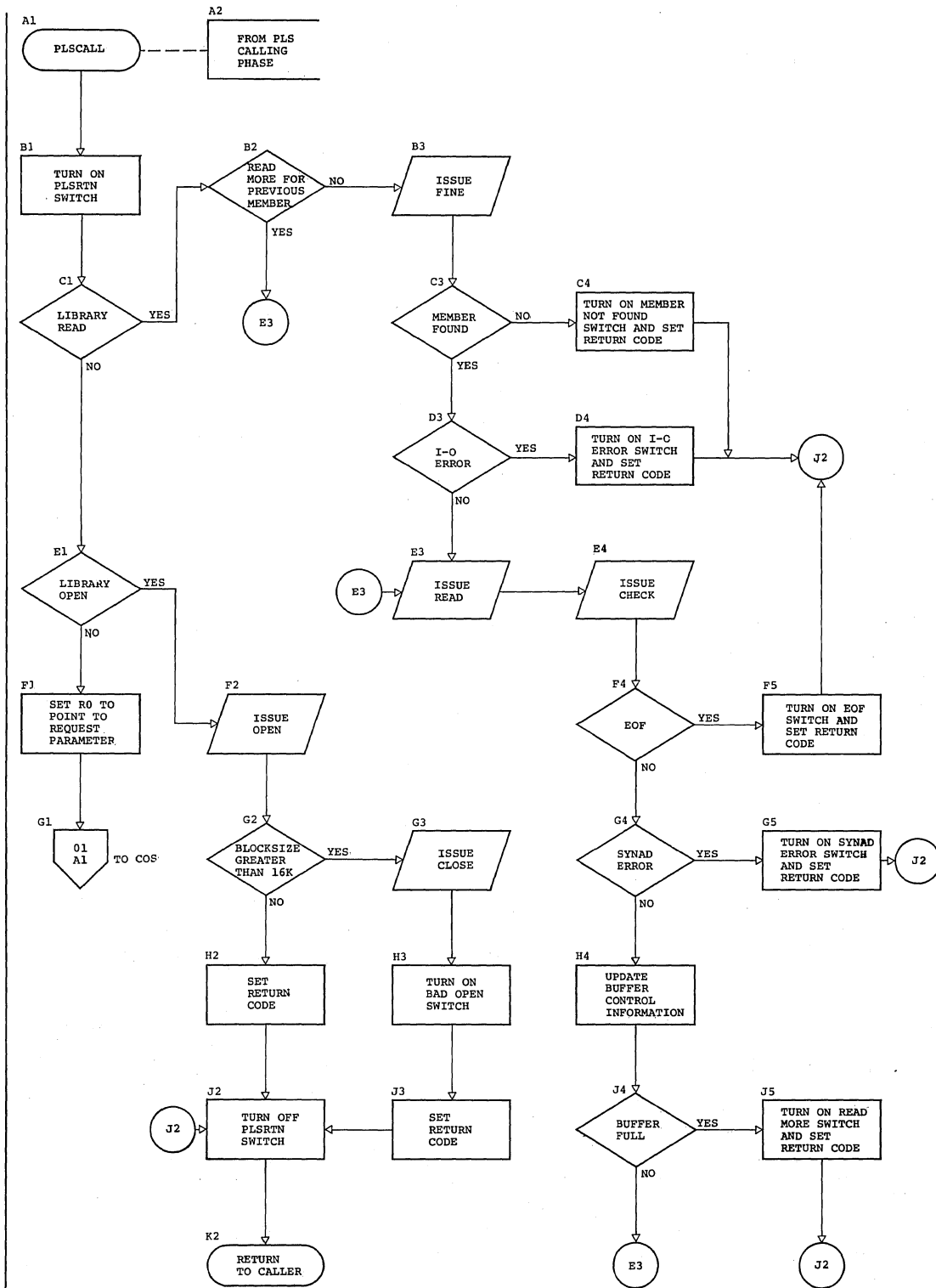


Chart BA. Phase 01: Overall Flow

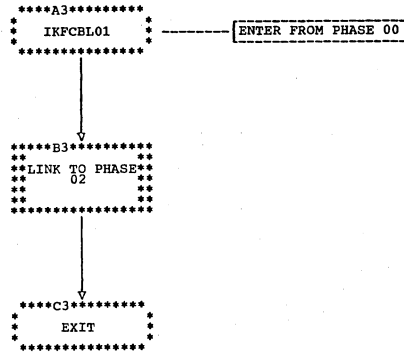
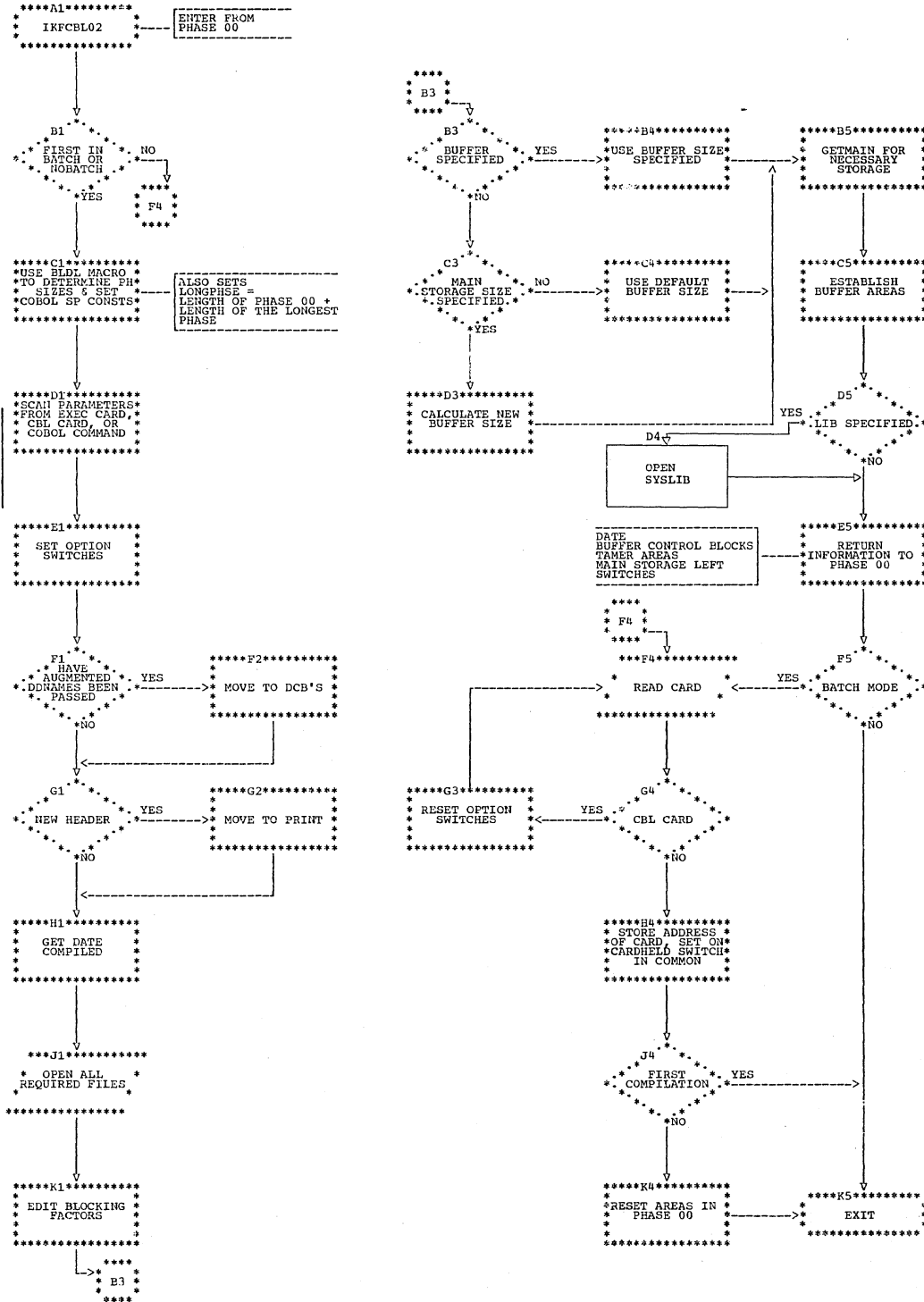
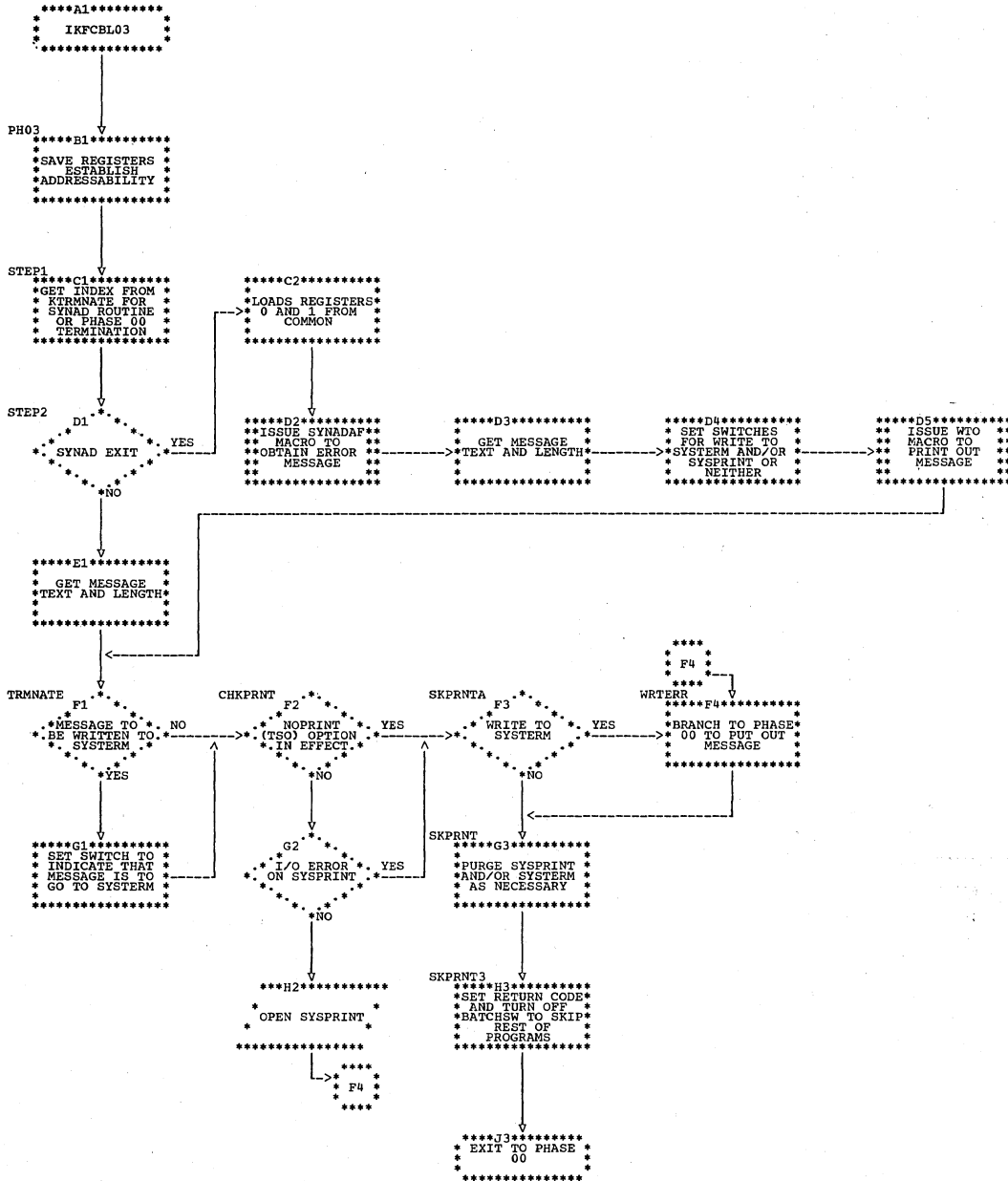


Chart BB. Phase 02: Overall



| Chart BC. Phase 03: Overall Flow





| Chart BD (Part 1 of 4). Phase 04: IKFCBL04

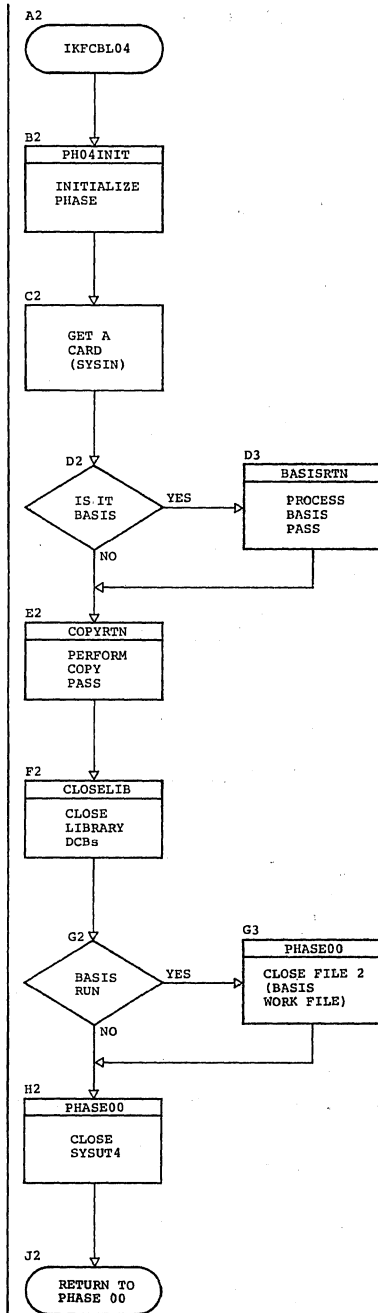


Chart BD (Part 2 of 4). Phase 04: BASISRTN

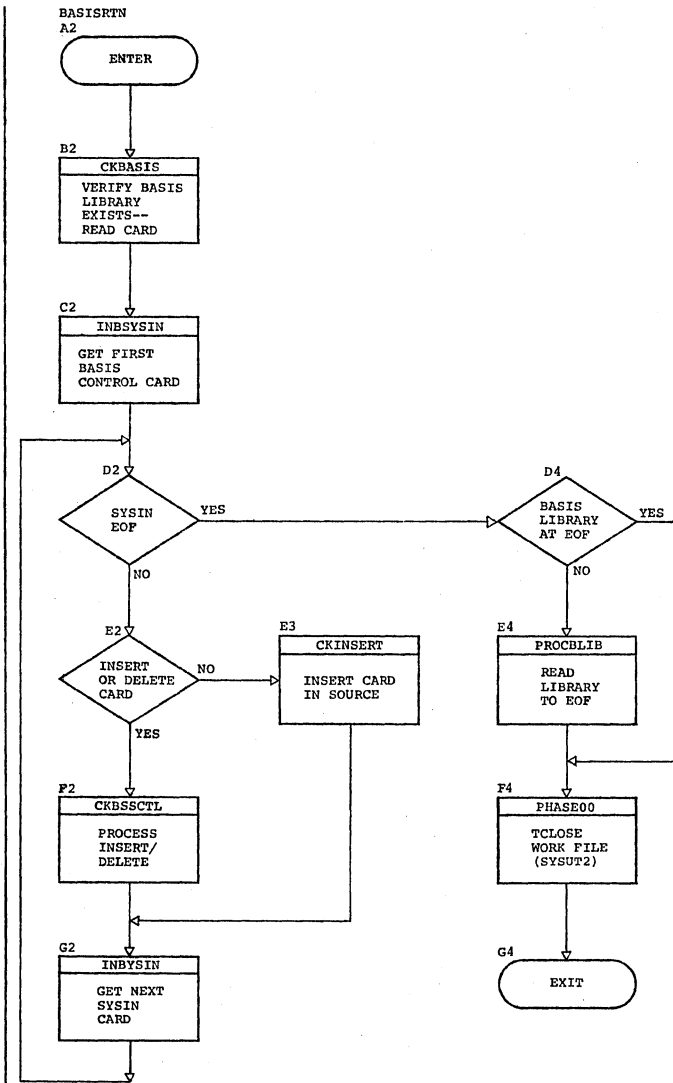


Chart BD (Part 3 of 4). Phase 04: COPYRTN

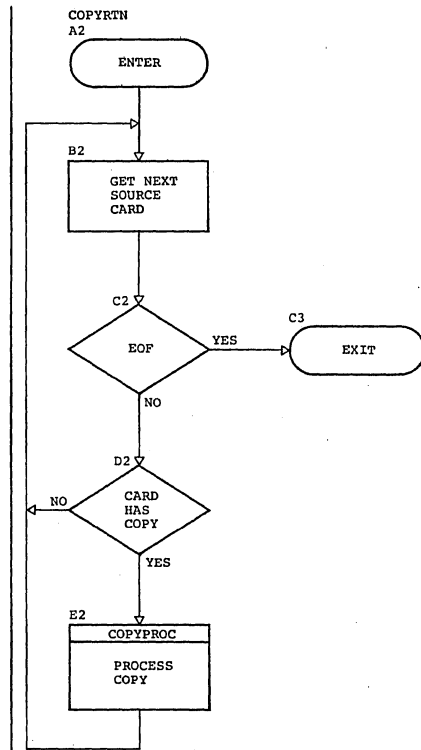
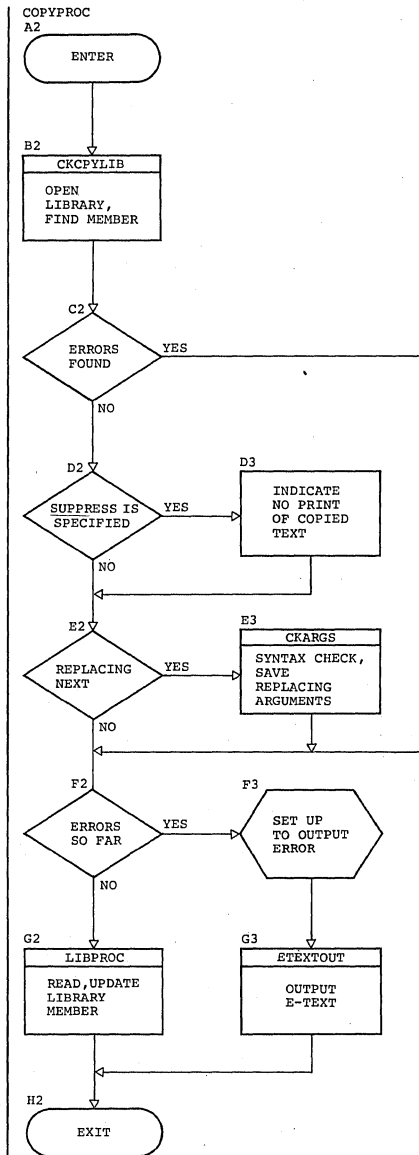


Chart BD (Part 4 of 4). Phase 04: COPYPROC



| Chart BE (Part 1 of 3). Phase 05: Overall Flow

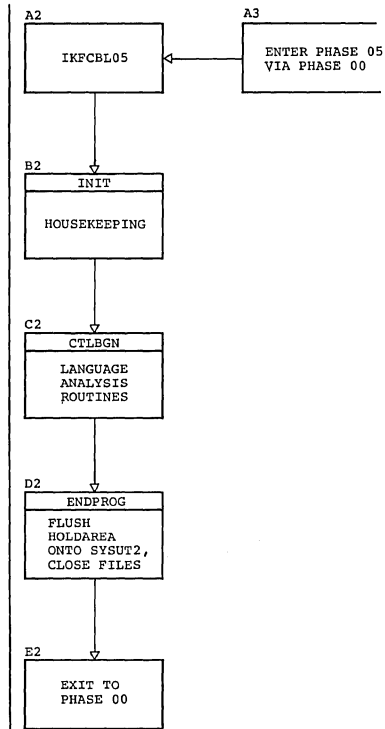


Chart BE (Part 2 of 3). Phase 05: Language Analysis Routine

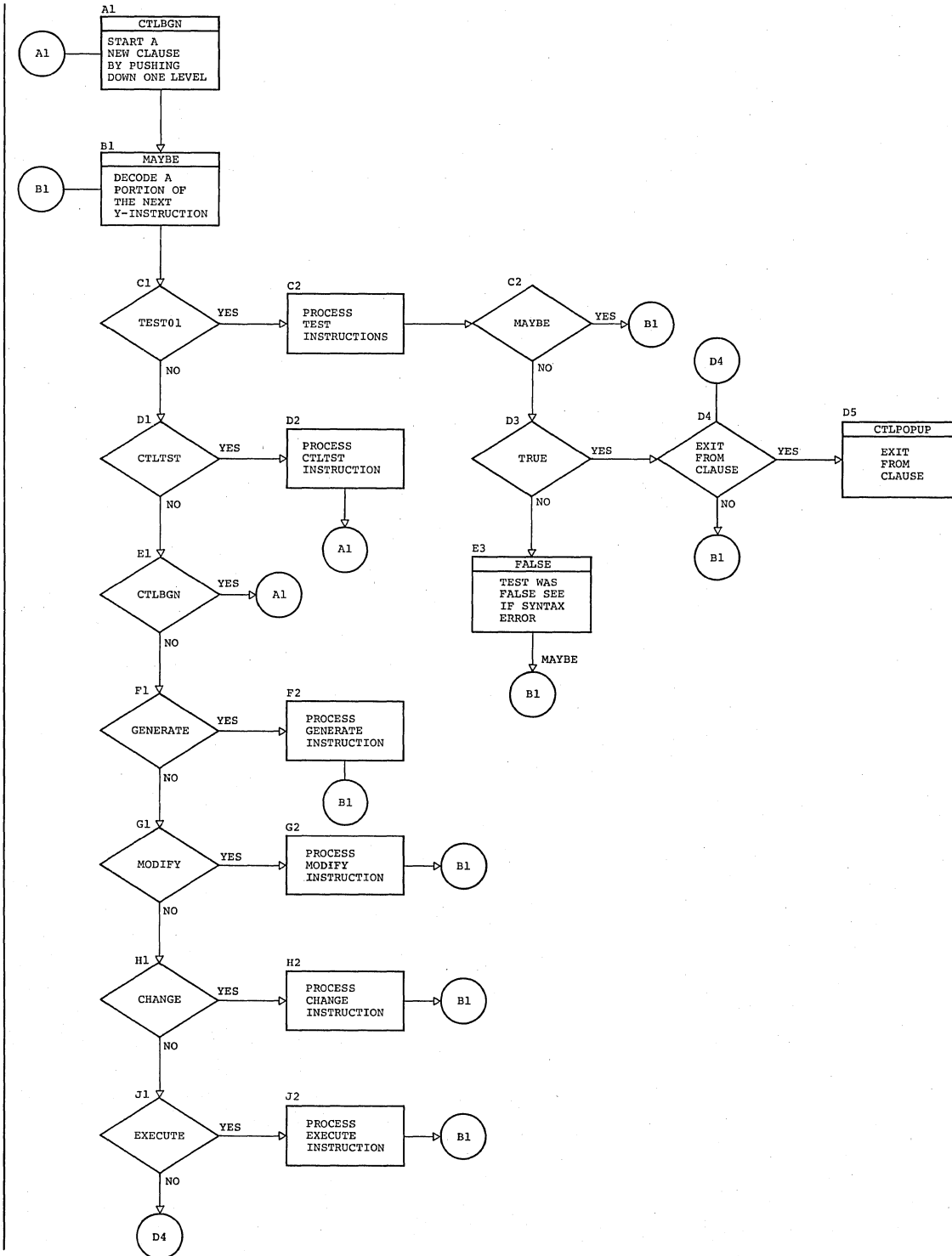


Chart BE (Part 3 of 3). Phase 05: Input and Scanning Routines (SCAN)

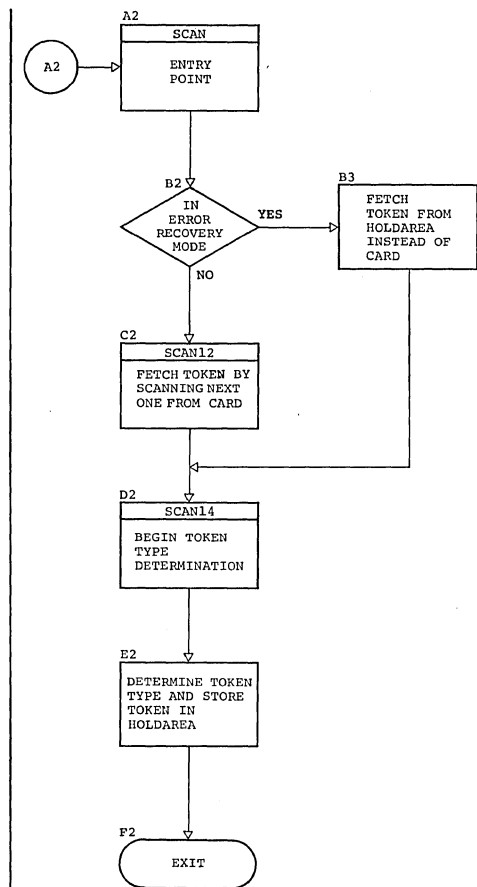


Chart BF (Part 1 of 4). Phase 06: Overall Flow

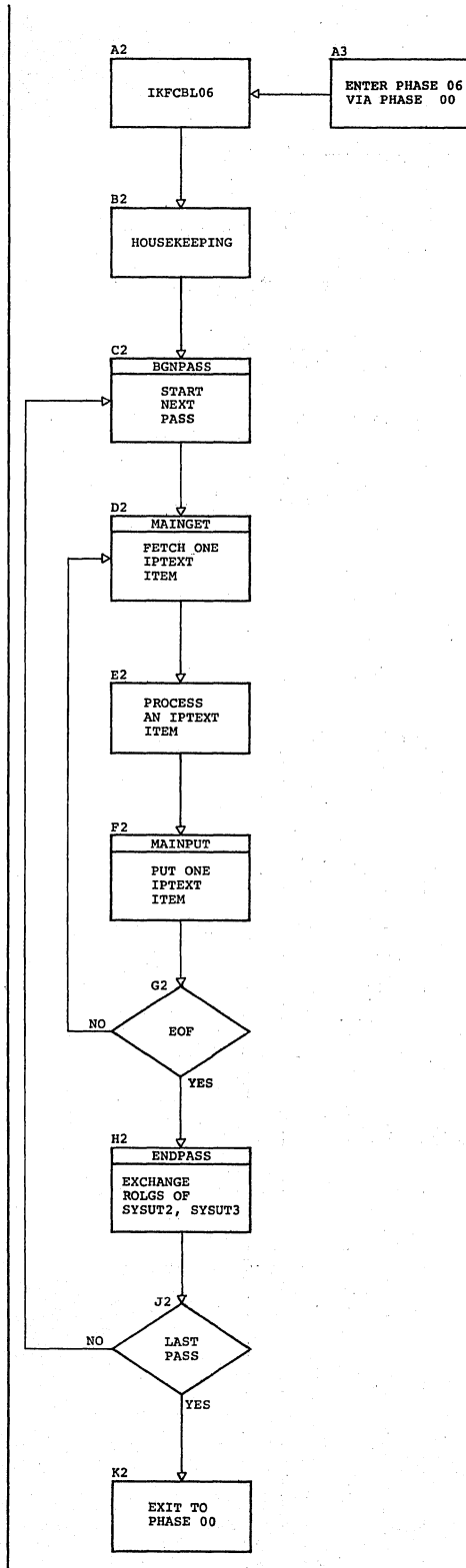




Chart BF (Part 2 of 4). Phase 06: IPTEXT ITEM Processors

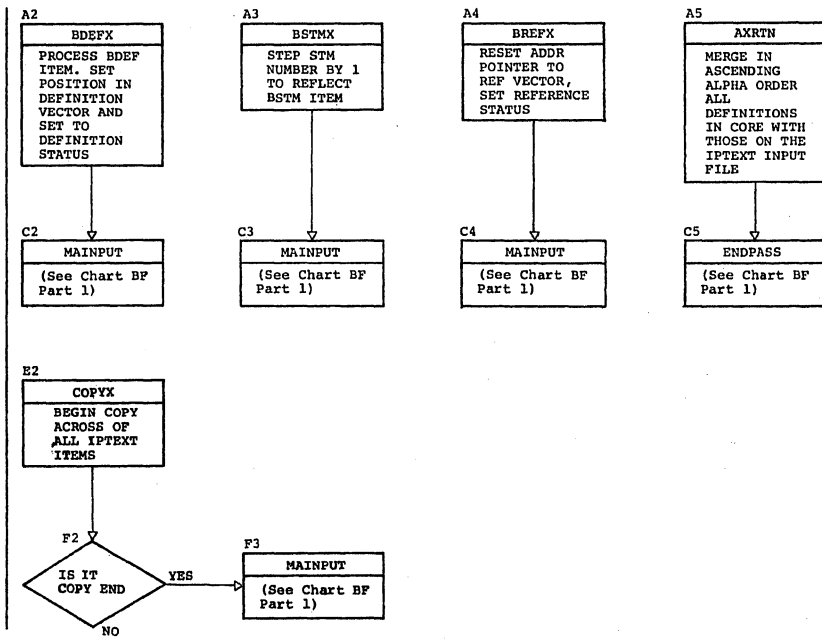
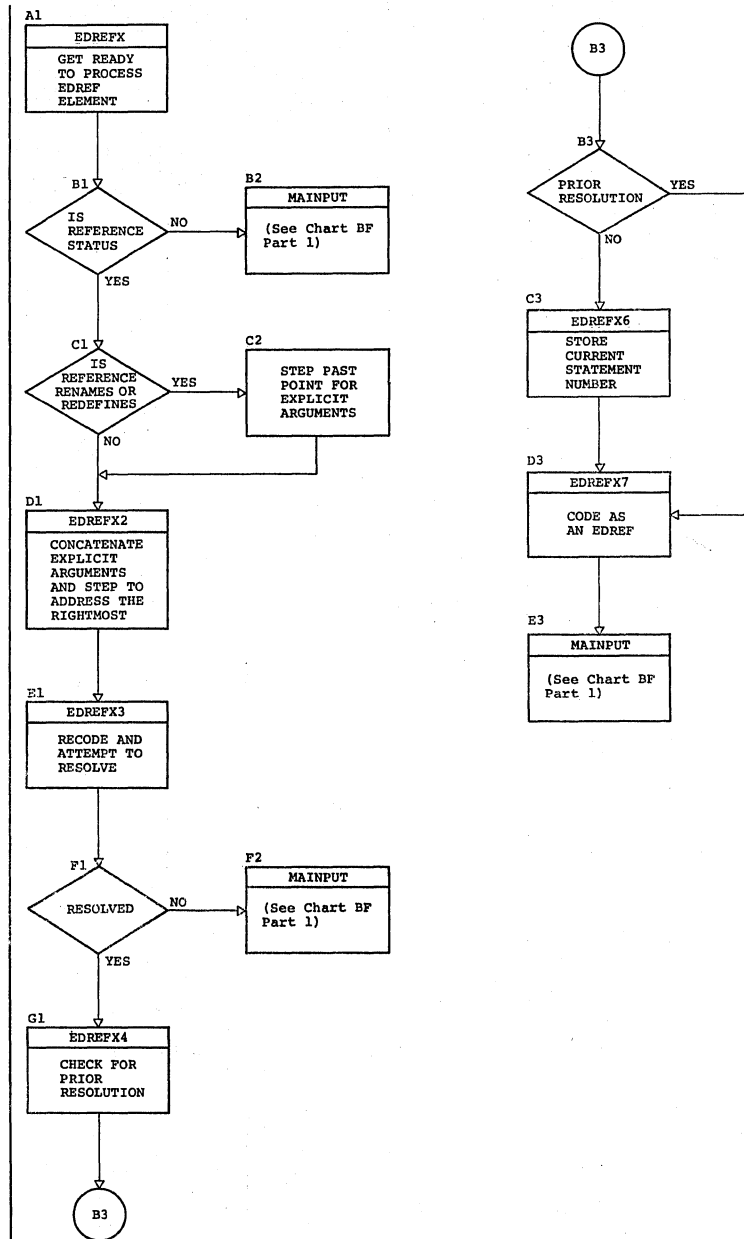


Chart BF (Part 3 of 4). Phase 06: IPTEXT ITEM Processors



1 Chart BF (Part 4 of 4). Phase 06: IPTEXT ITEM PROCESSORS

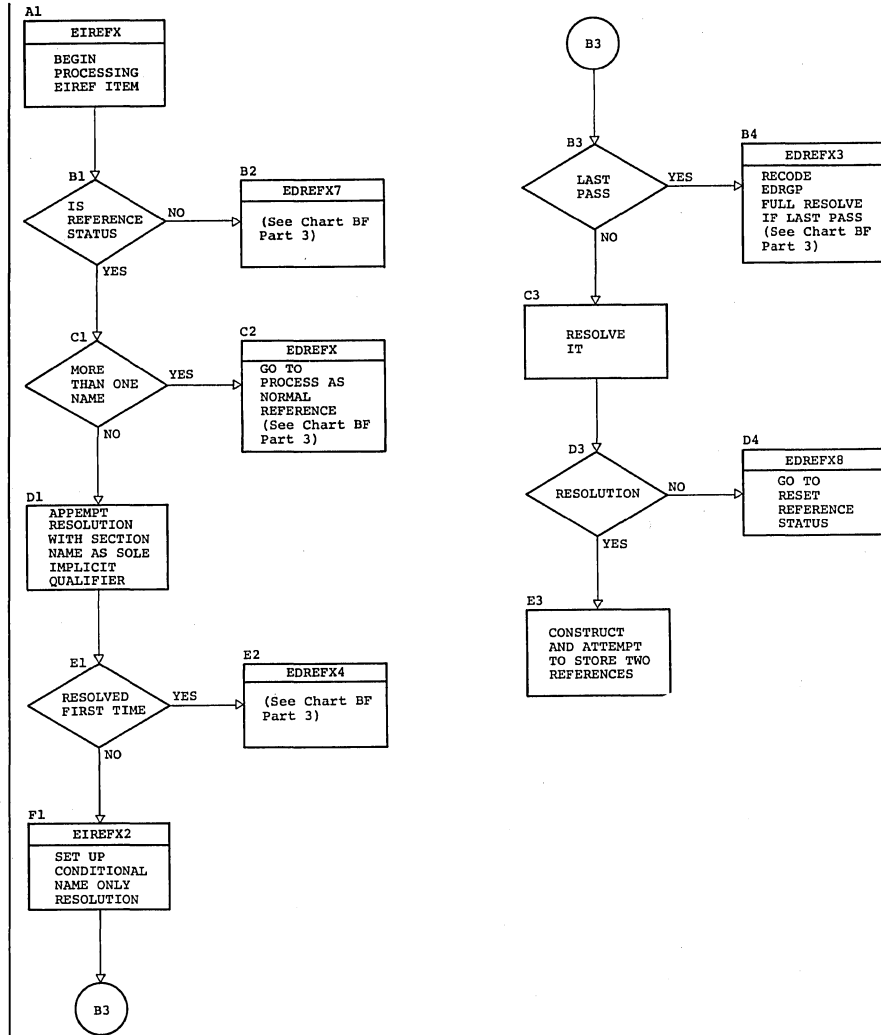
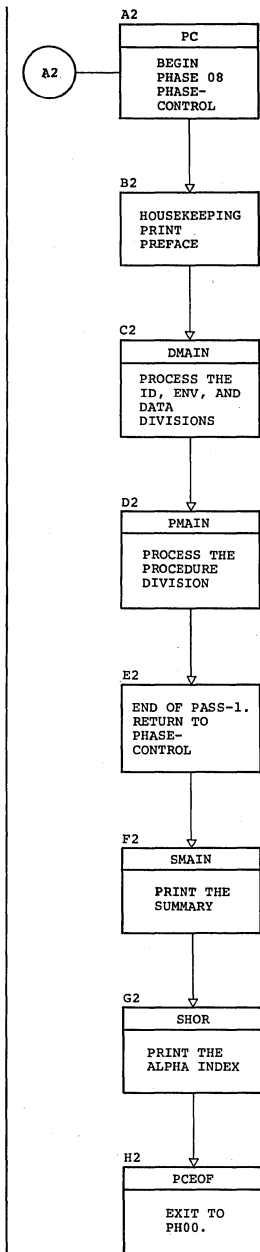


Chart BG (Part 1 of 2). Phase 08: Overall Flow



| Chart BG (Part 2 of 2). Phase 08: Data Division Flow

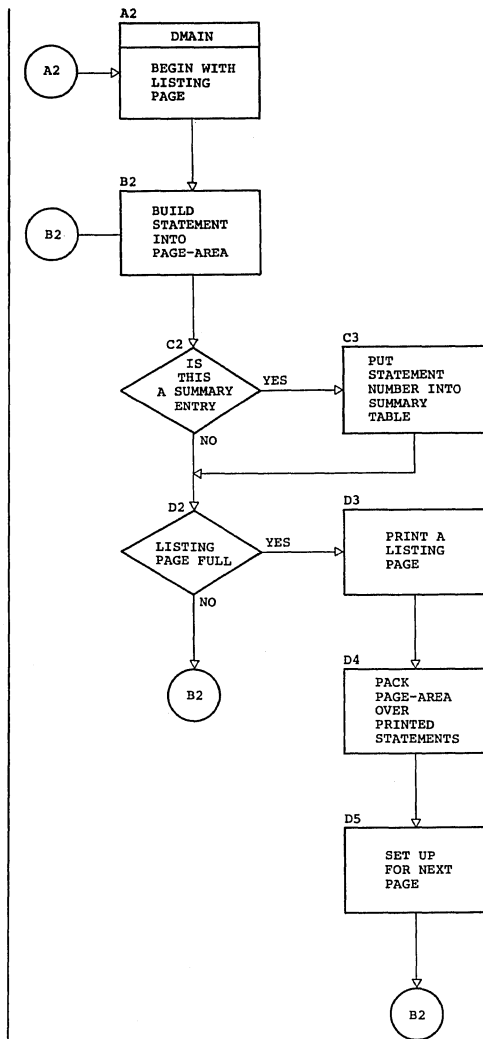


Chart CA. Phase 10: Overall Flow

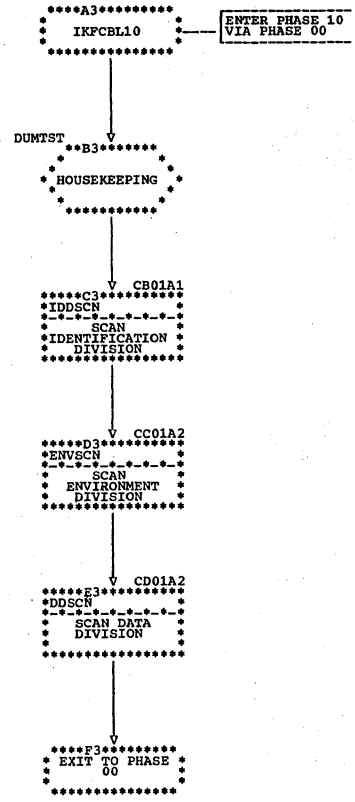


Chart CB. Phase 10: IDDCSN Routine

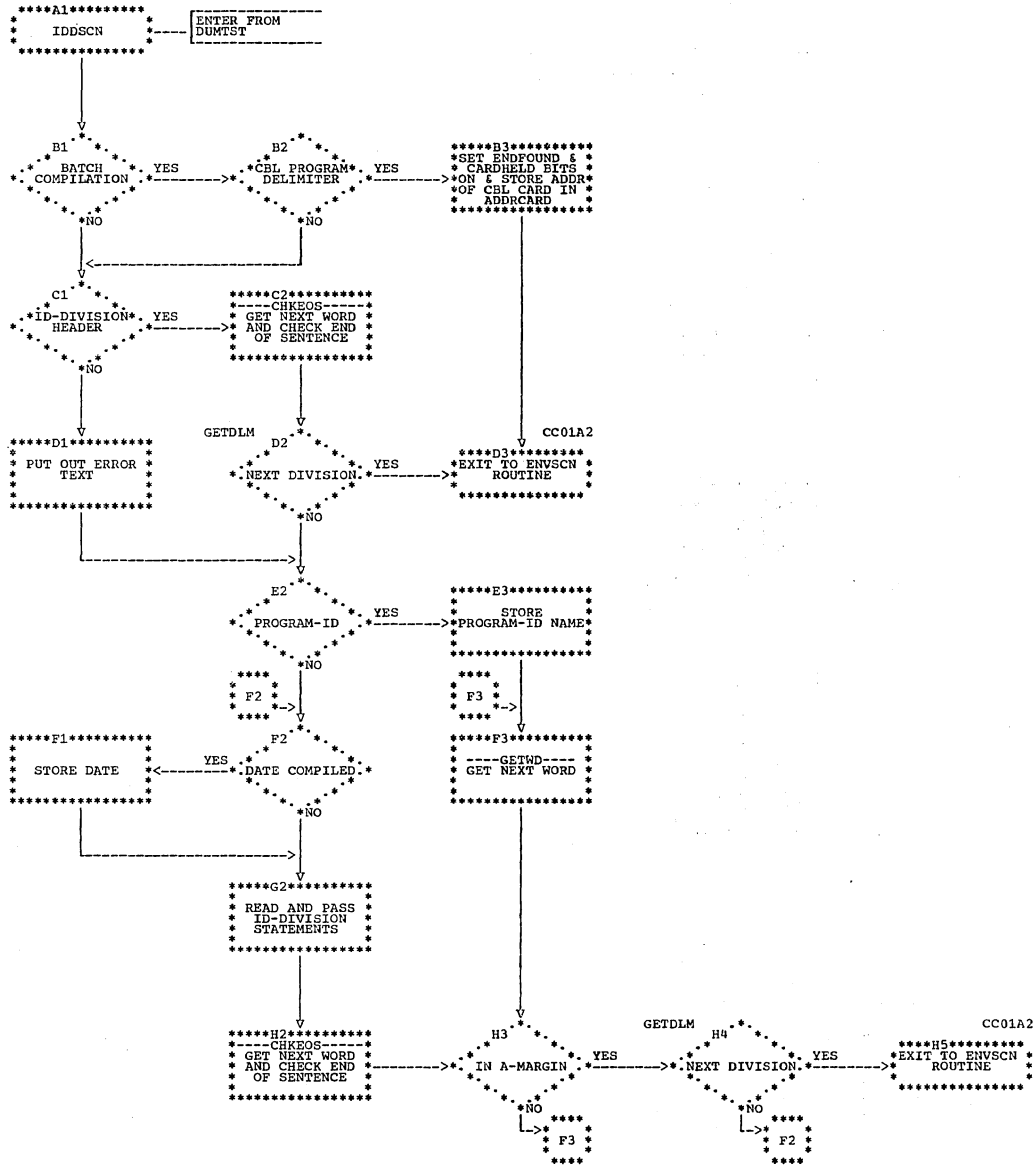


Chart CC. Phase 10: ENVSCN Routine

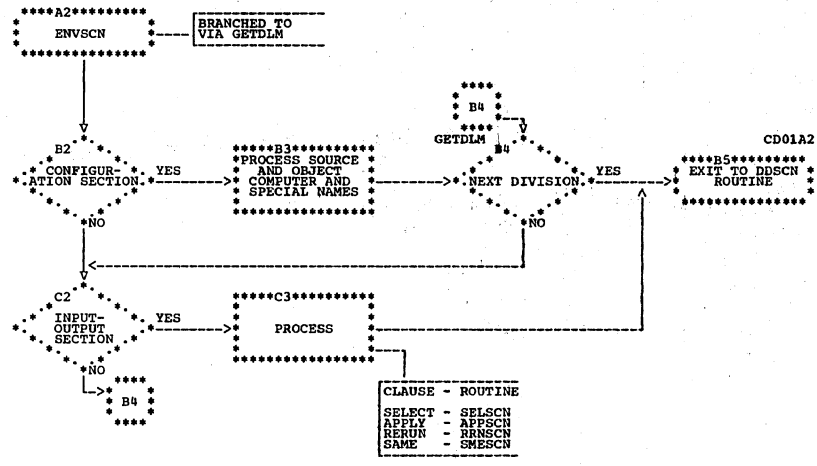




Chart CD. Phase 10: DDSCN Routine

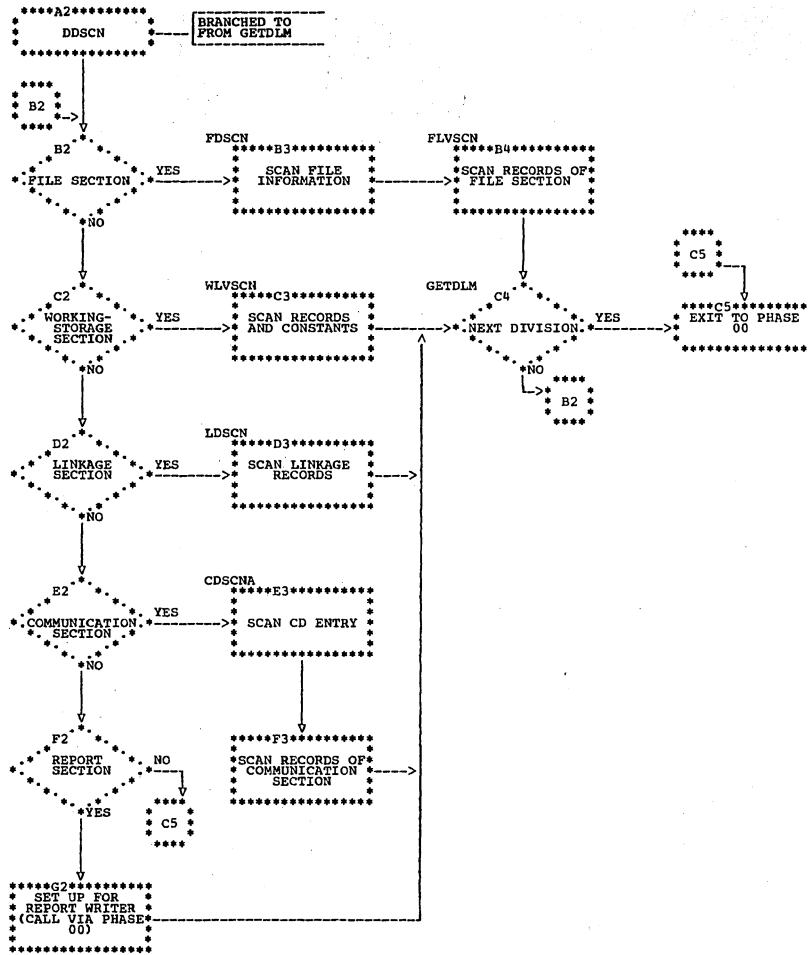
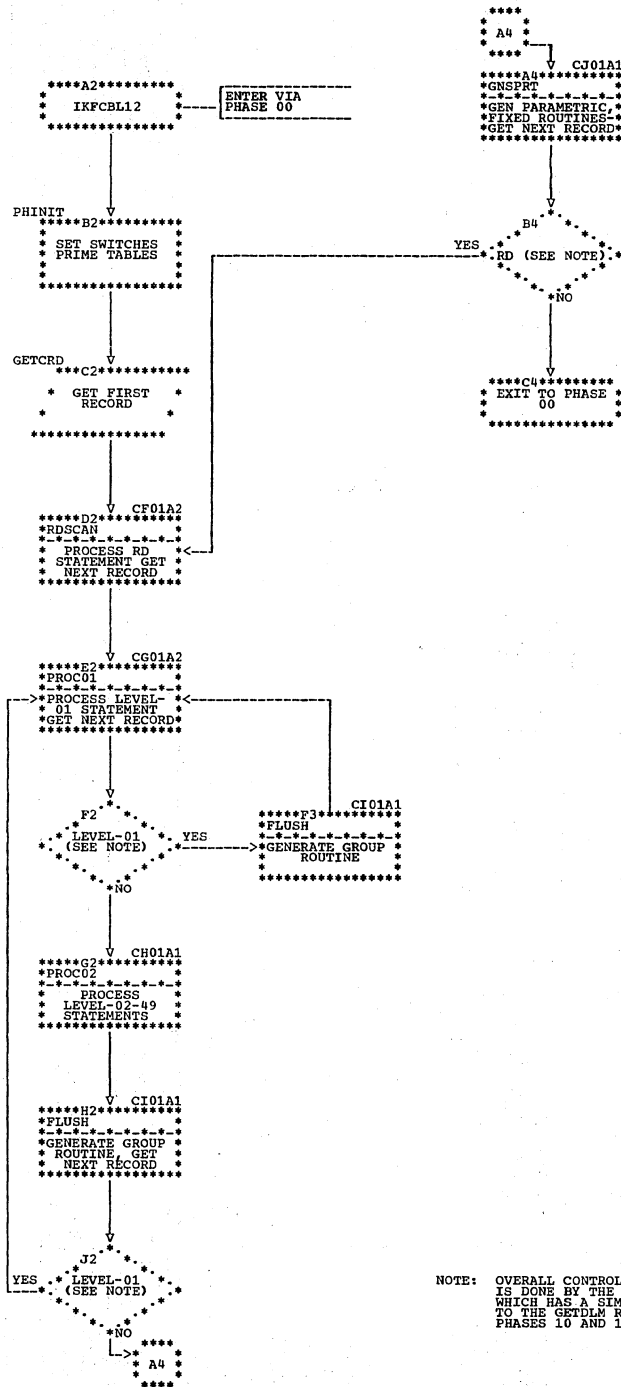


Chart CE. Phase 12: Overall Flow



NOTE: OVERALL CONTROL OR PROCESSING IS DONE BY THE GETDLN ROUTINE, WHICH HAS A SIMILAR FUNCTION TO THE GETDLN ROUTINE IN PHASES 10 AND 1B

Chart CF. Phase 12: RDSCAN Routine

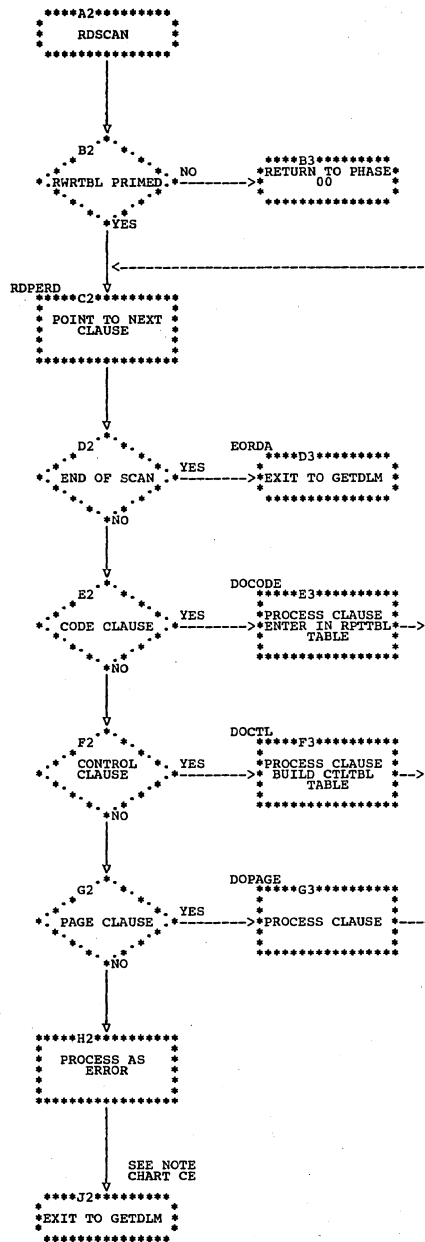


Chart CG. Phase 12: PROC01 Routine

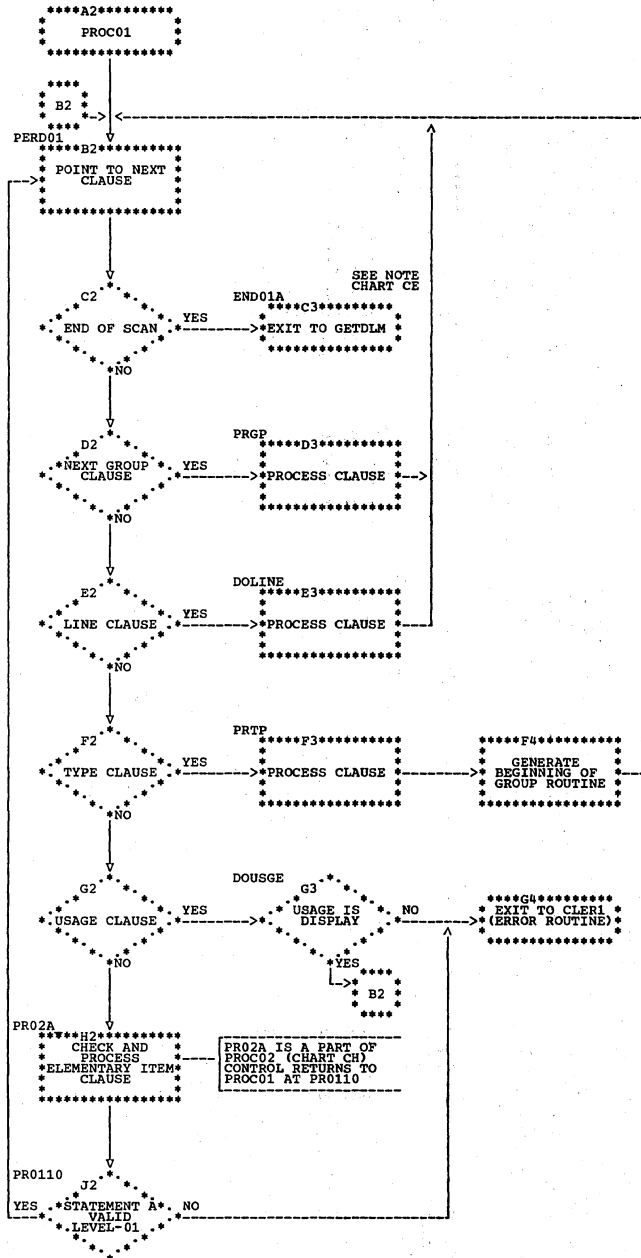


Chart CH. Phase 12: PROC02 Routine

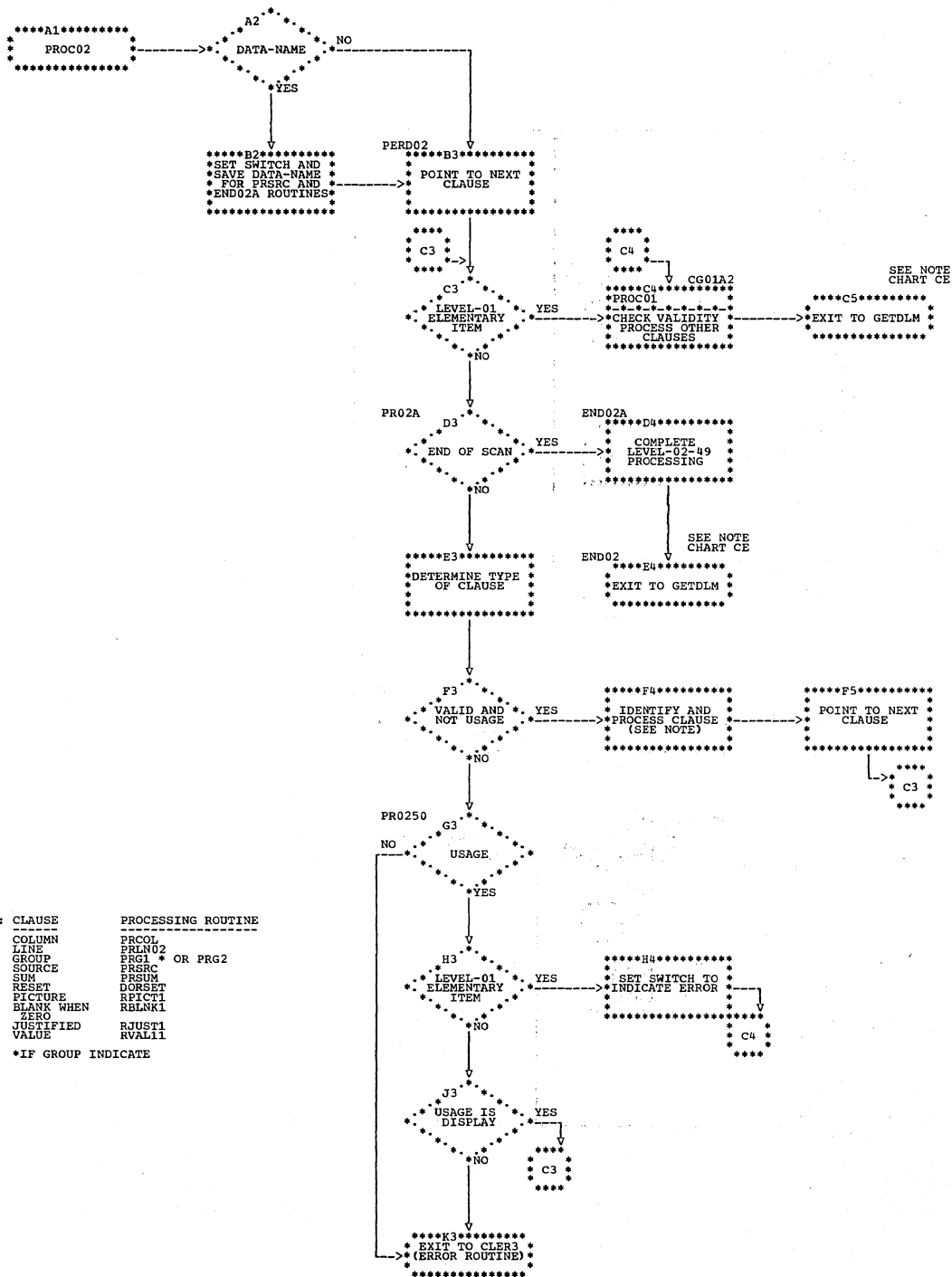


Chart CI. Phase 12: FLUSH Routine

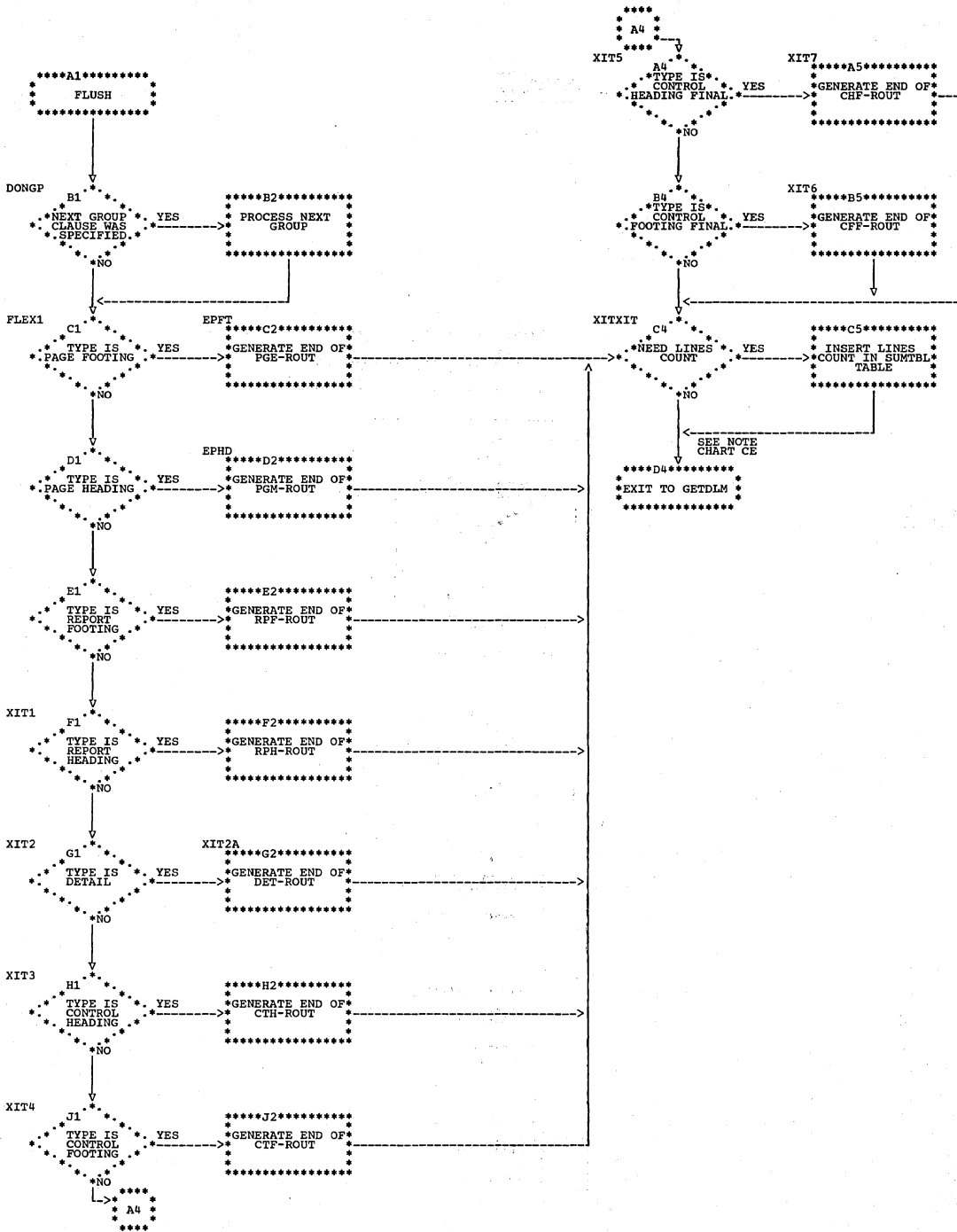


Chart CJ. Phase 12: GNSPRT and SPCRTS Routines

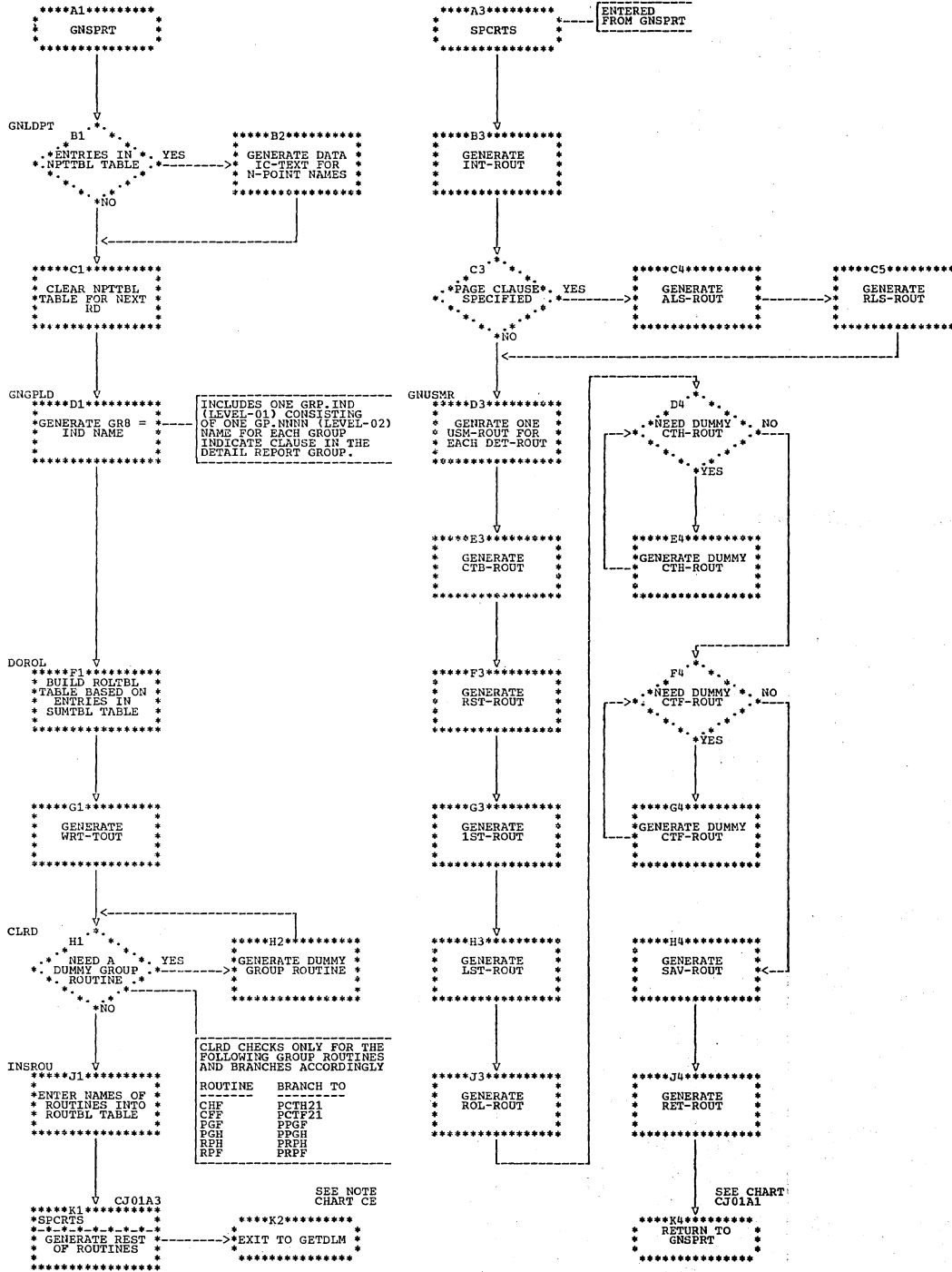


Chart CK. Phase 1B: Overall Flow (PDSCN Routine)

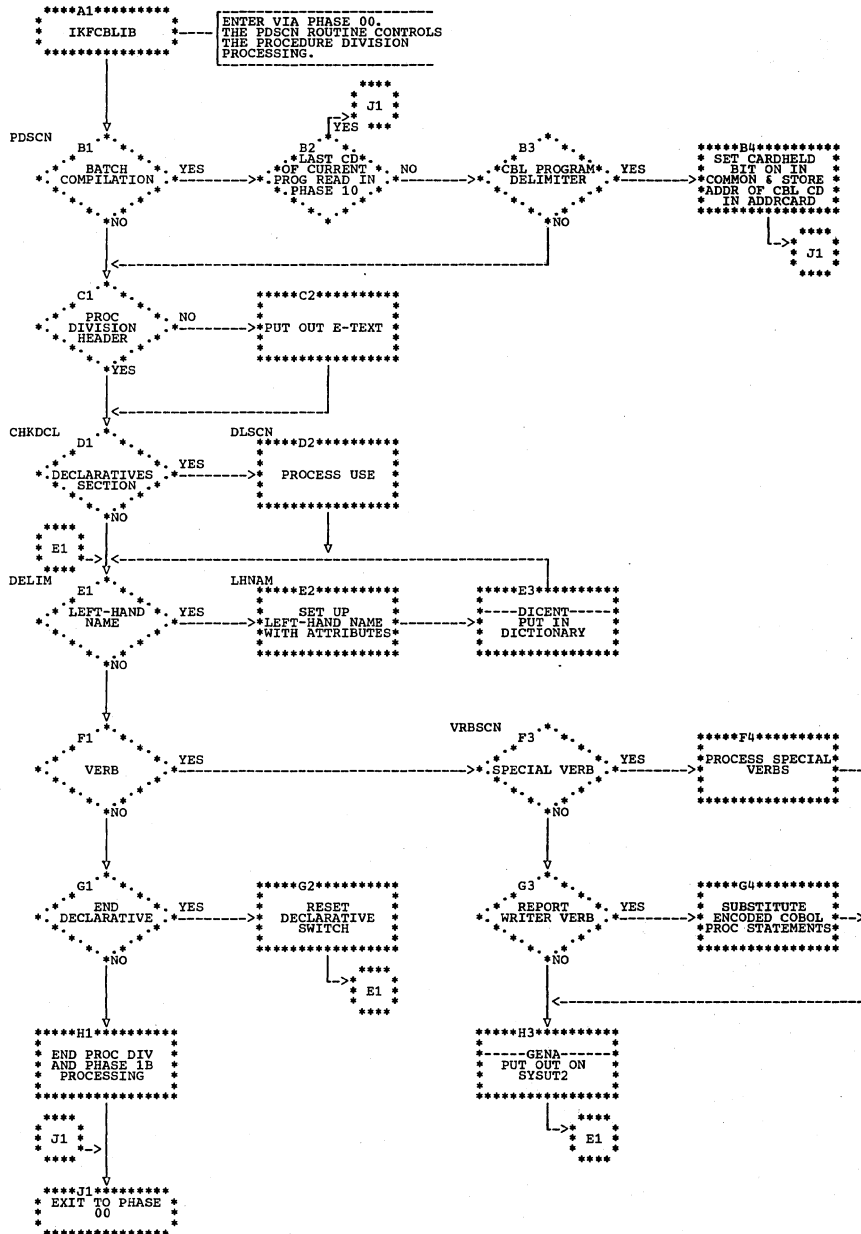




Chart DA. Phase 20: Overall Flow

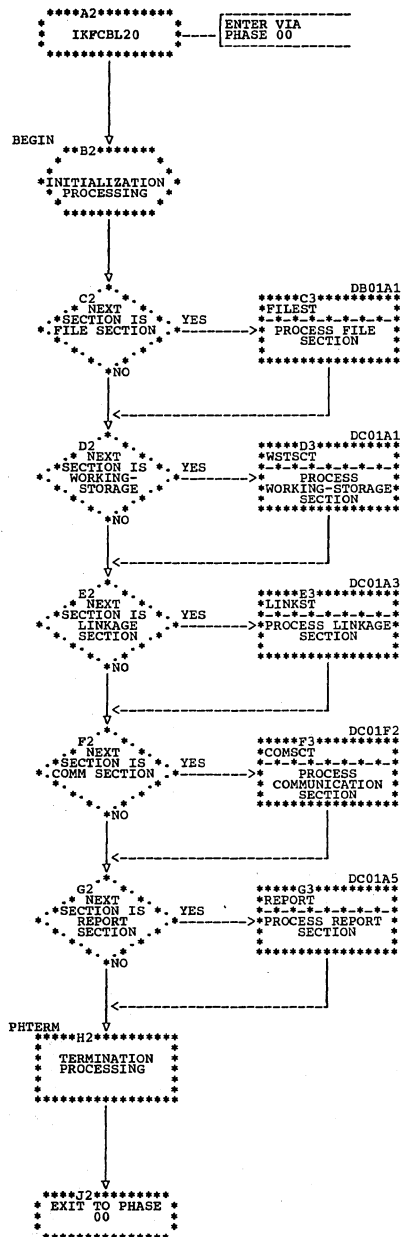


Chart DB. Phase 20: FILEST Routine

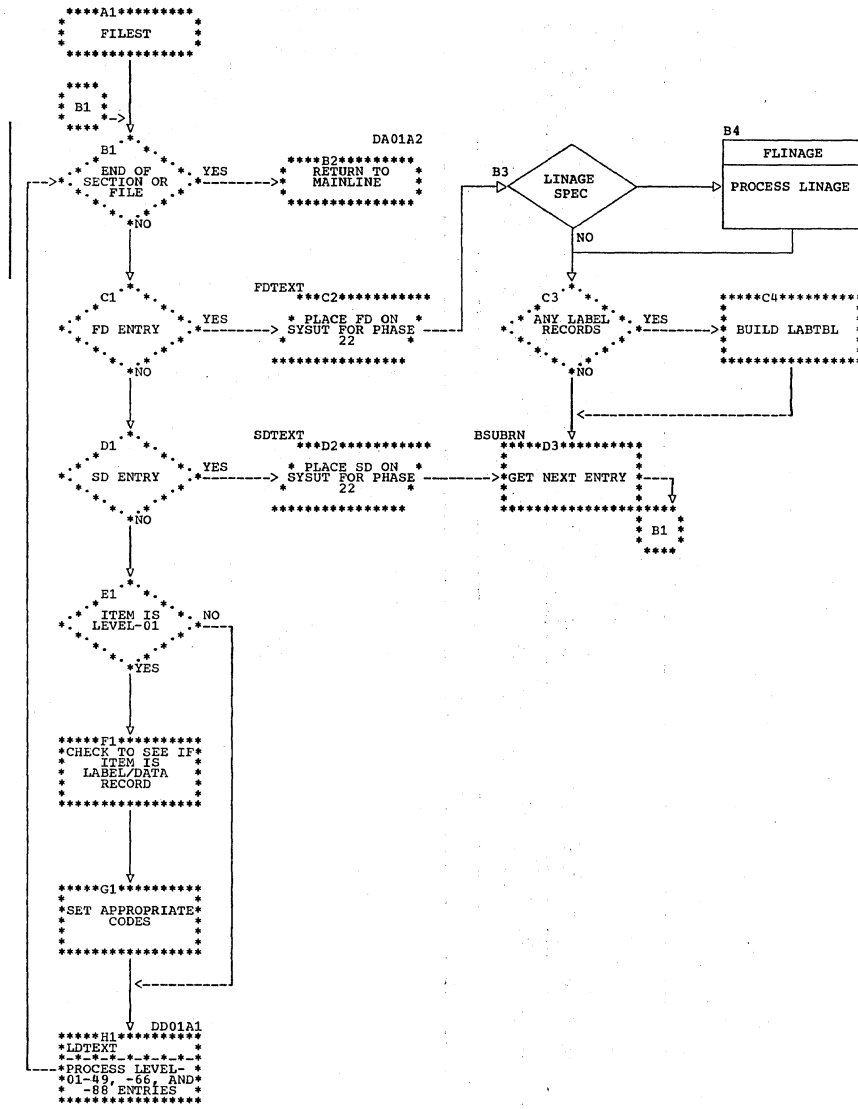


Chart DC. Phase 20: WSTSTCT, LINKST, COMSCT, and REPORT Routines

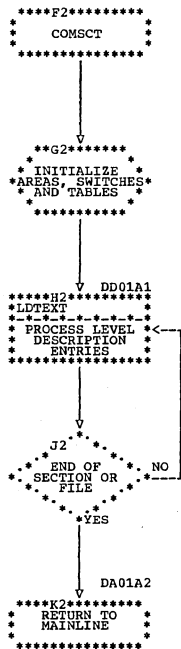
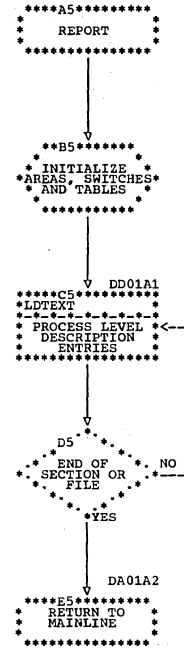
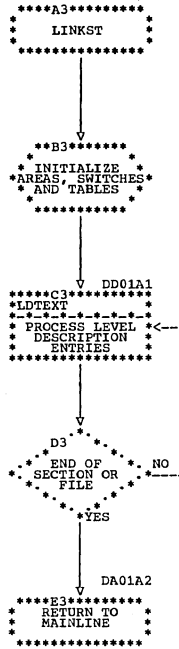
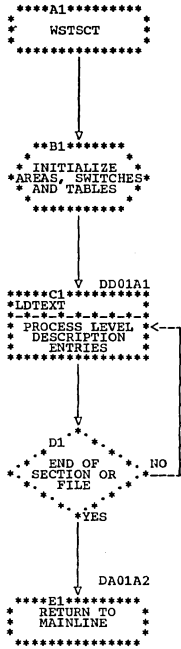


Chart DD. Phase 20: LDTEXT Routine

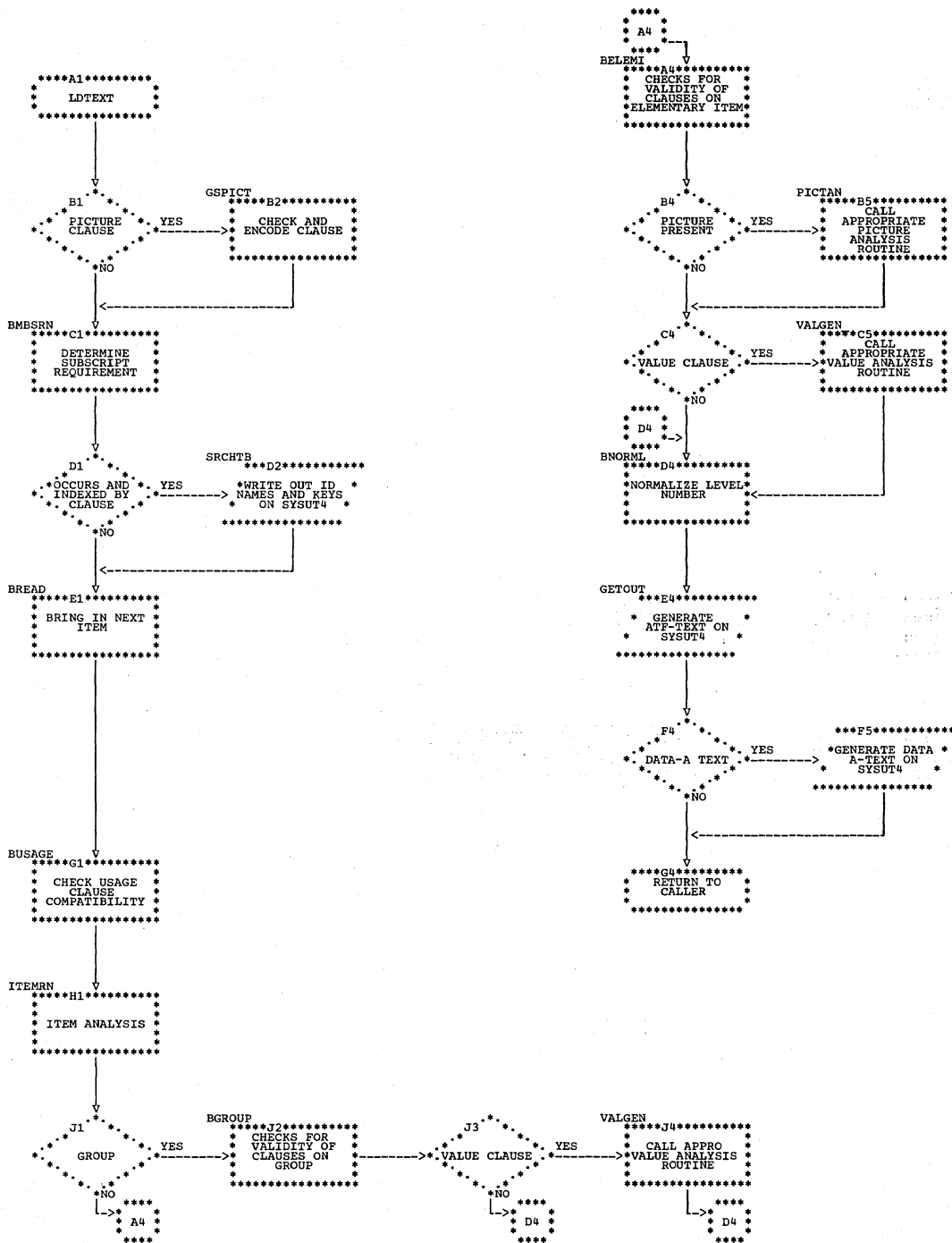


Chart DE. Phase 22: Overall Flow

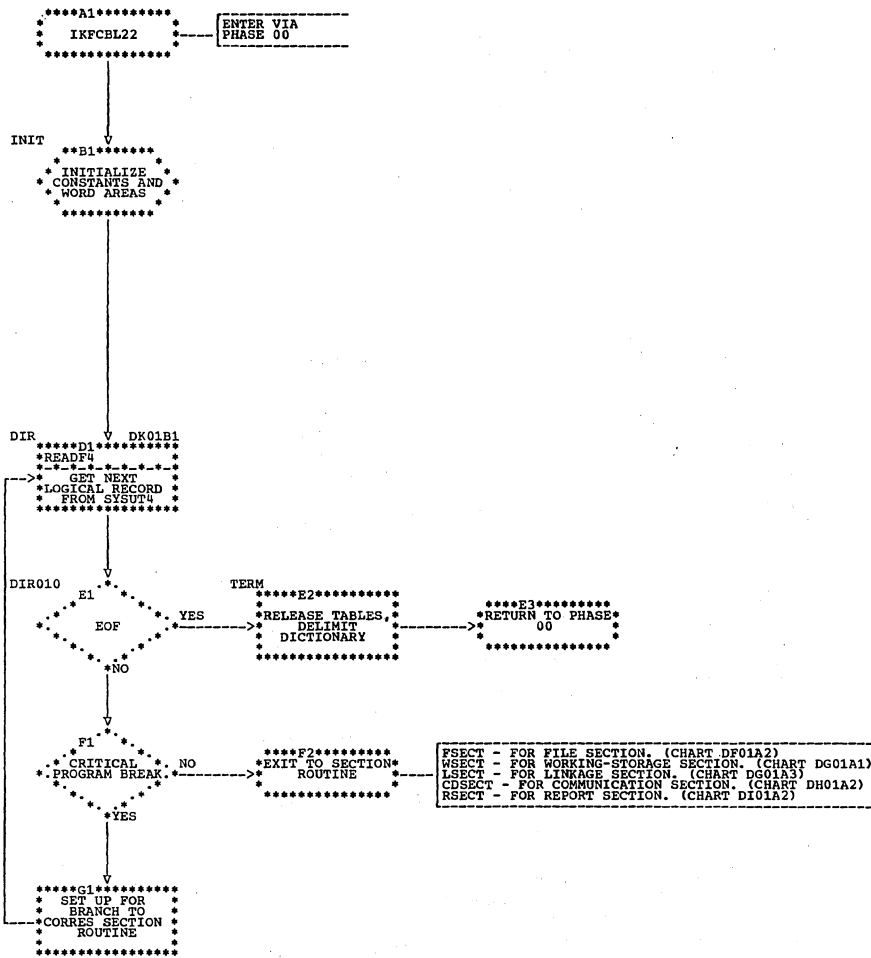


Chart DF. Phase 22: FSECT Routine

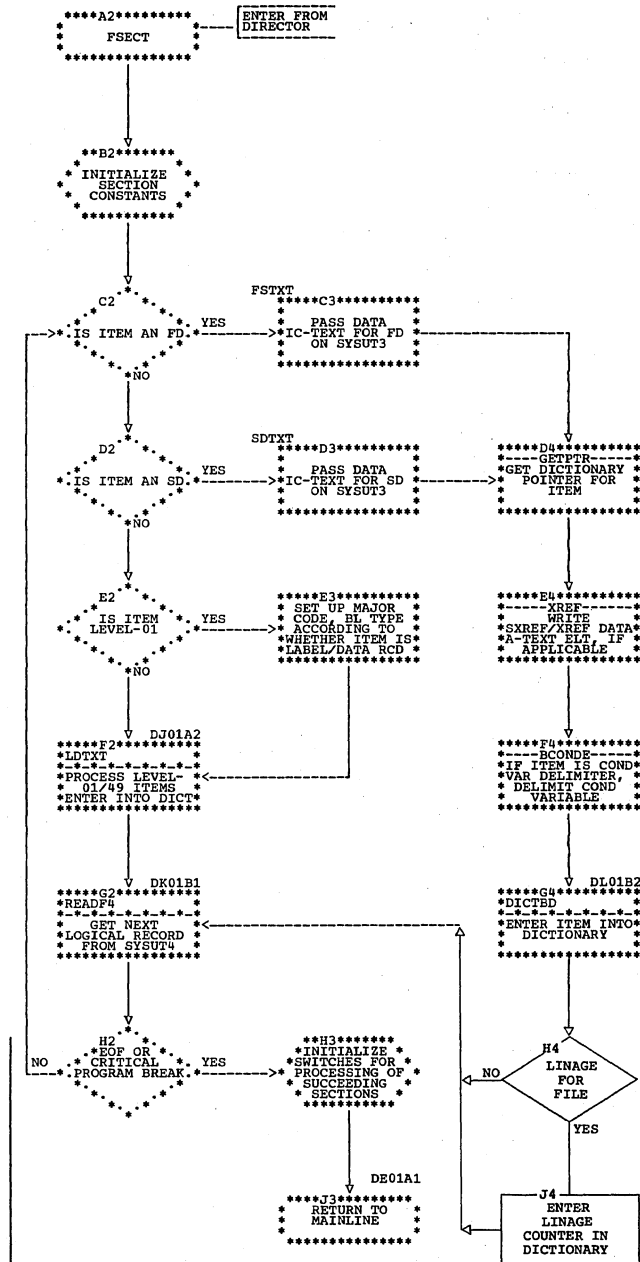


Chart DG. Phase 22: WSECT and LSECT Routines

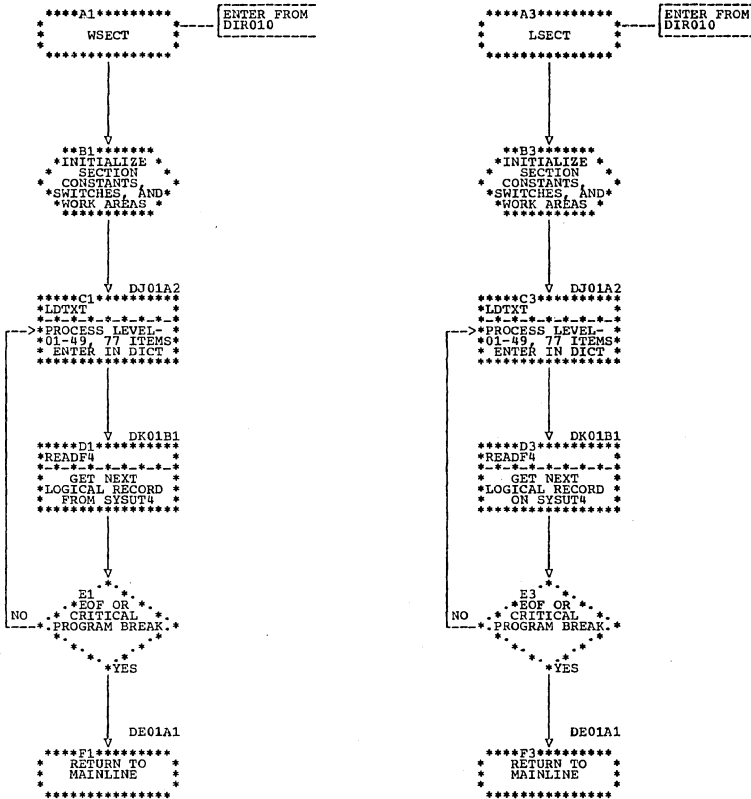


Chart DH. Phase 22: CDSECT Routine

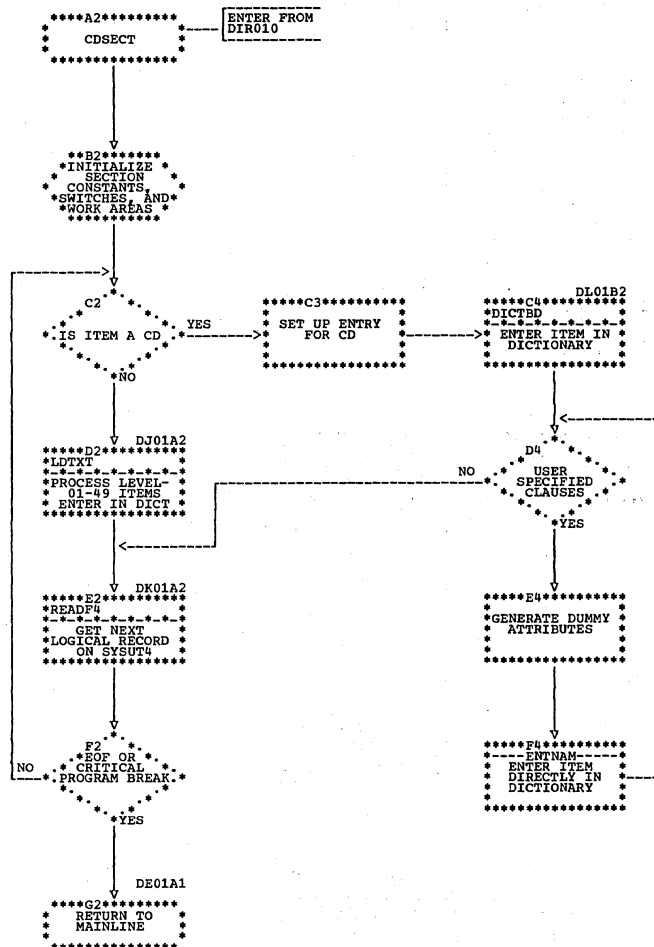




Chart DI. Phase 22: RSECT Routine

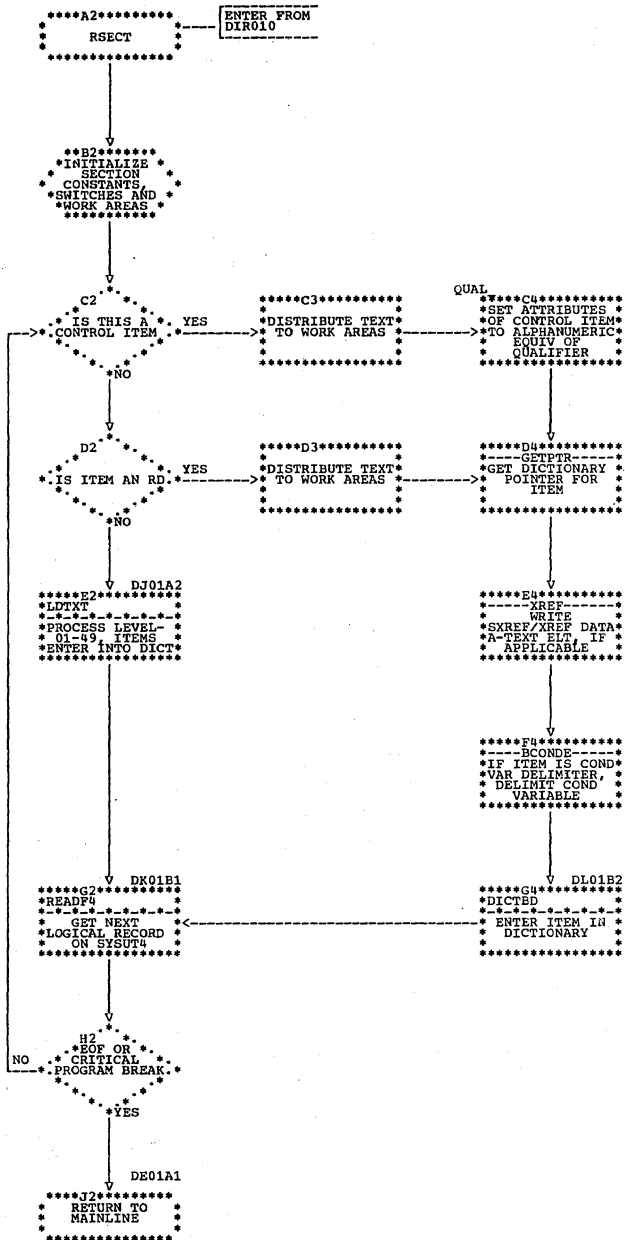


Chart DJ. Phase 22: LDTXT Routine

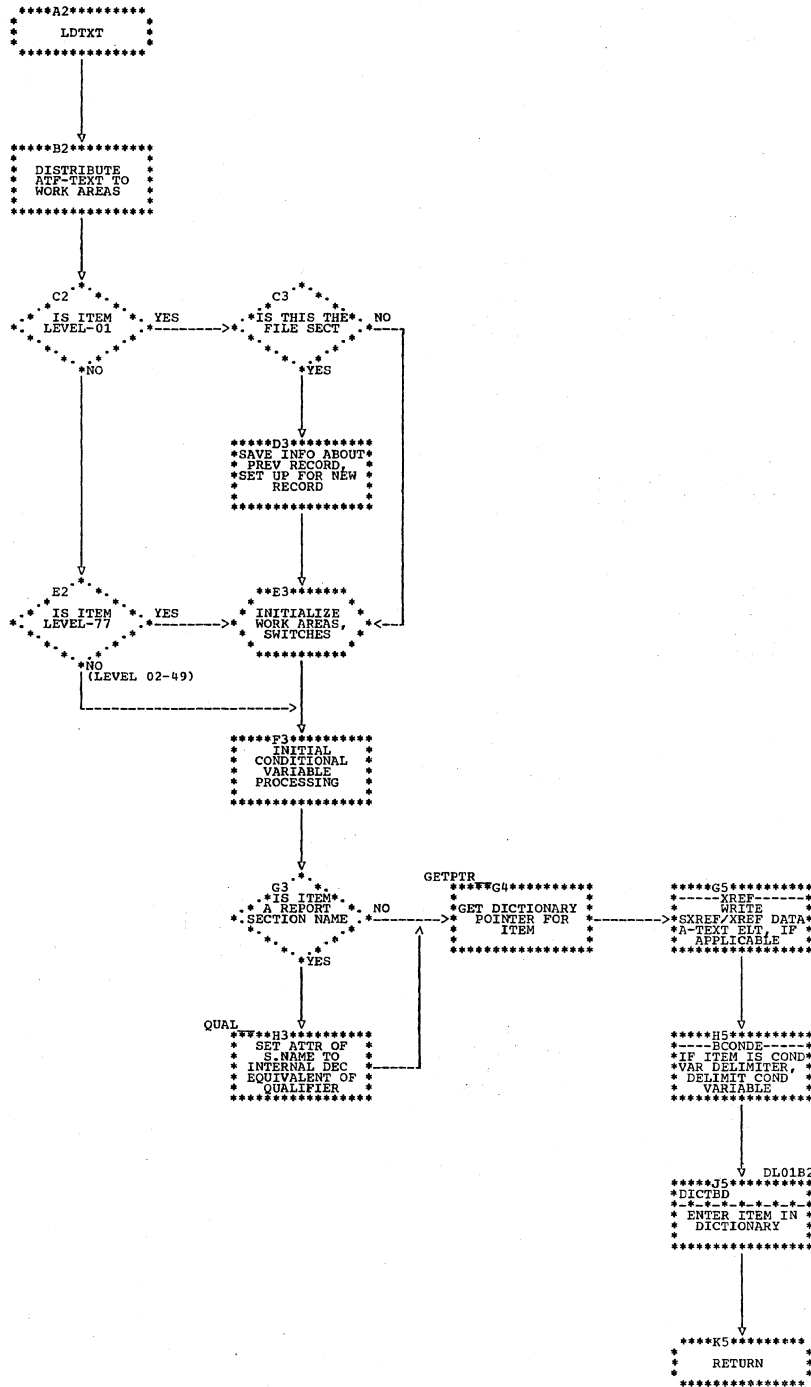


Chart DK. Phase 22: READF4 Routine

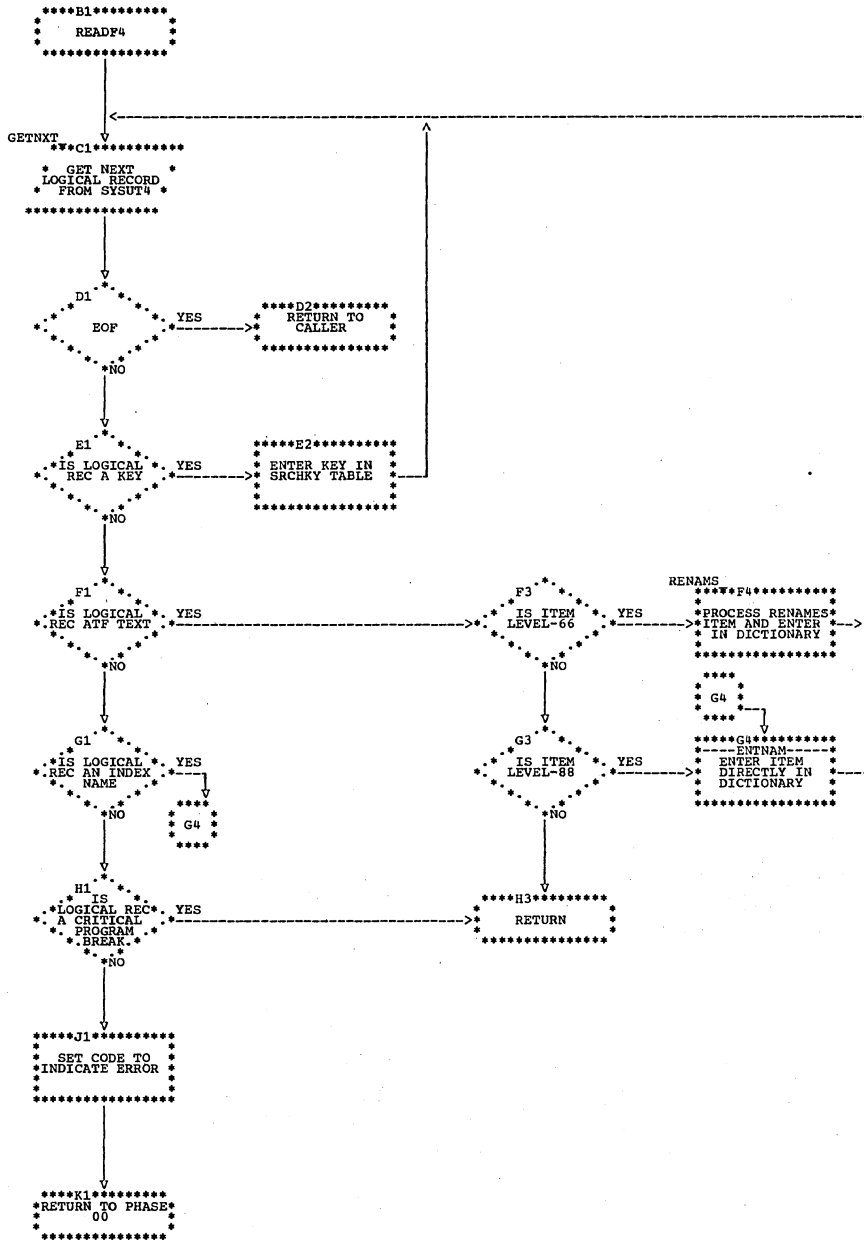


Chart DL. Phase 22: DICTBD Routine

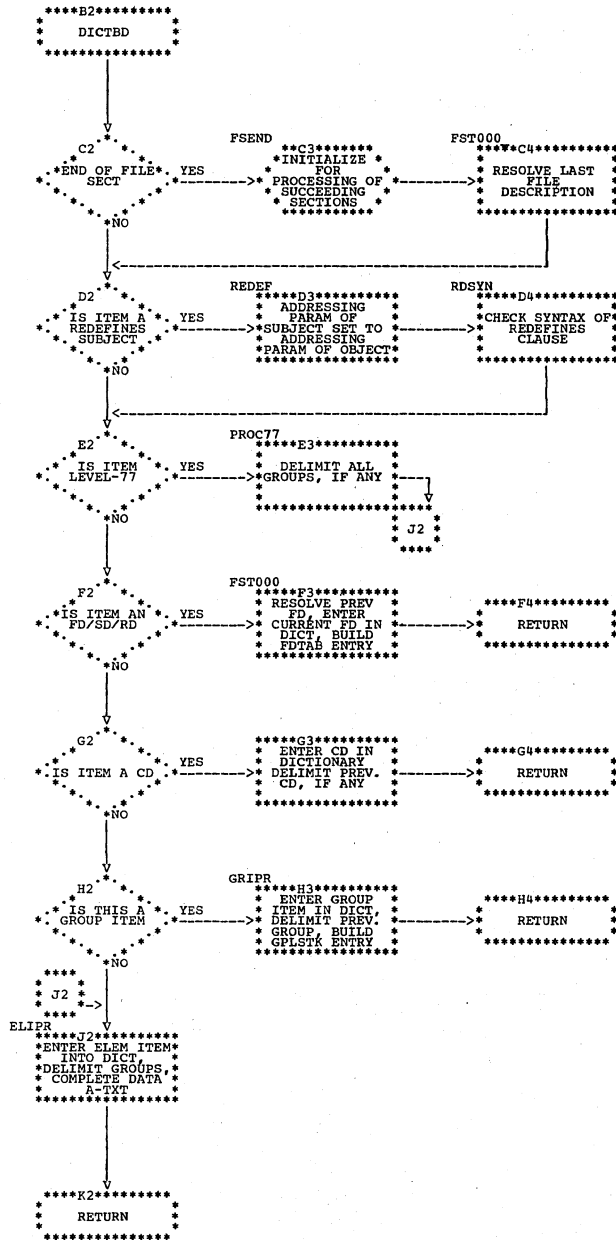


Chart DM. Phase 21: Overall Flow

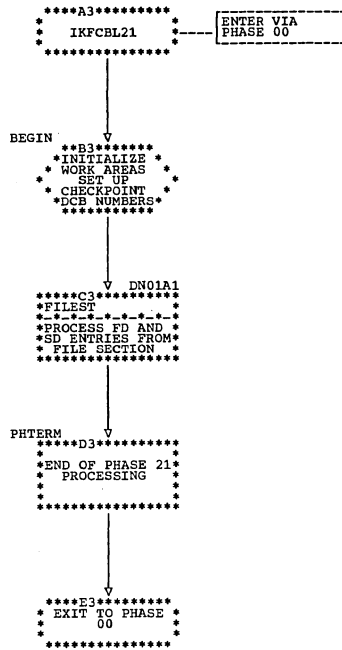


Chart DN. Phase 21: FILEST Routine

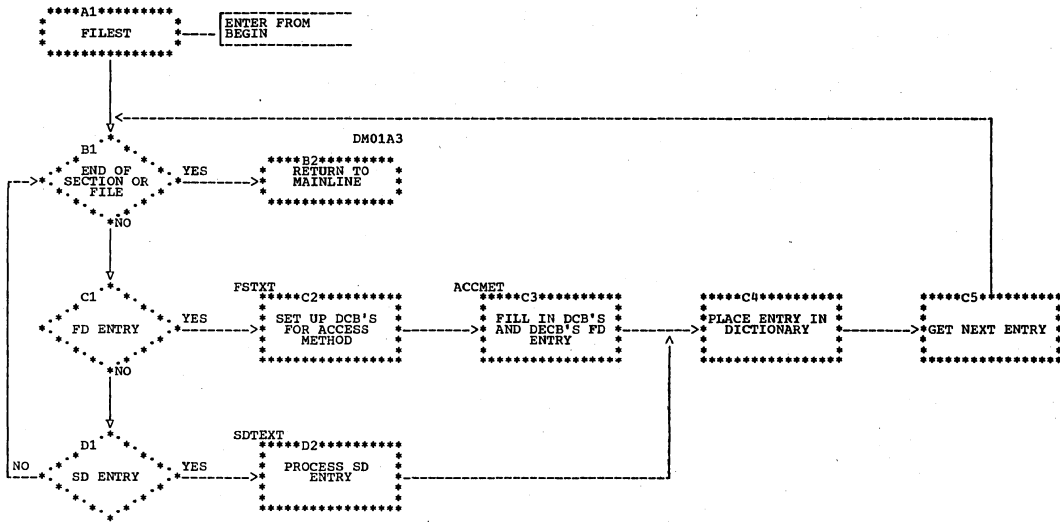


Chart D0. Phase 25: Overall Flow

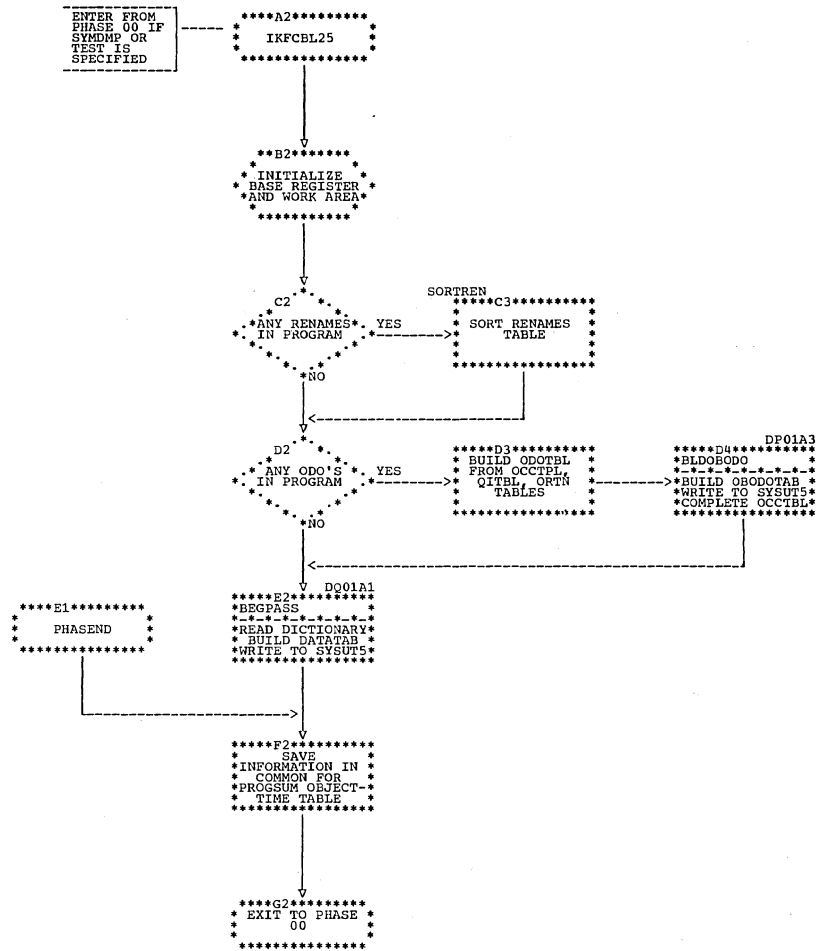


Chart DP. Phase 25: ODOBLD, BLDOBODO, and ENDP1 Routines

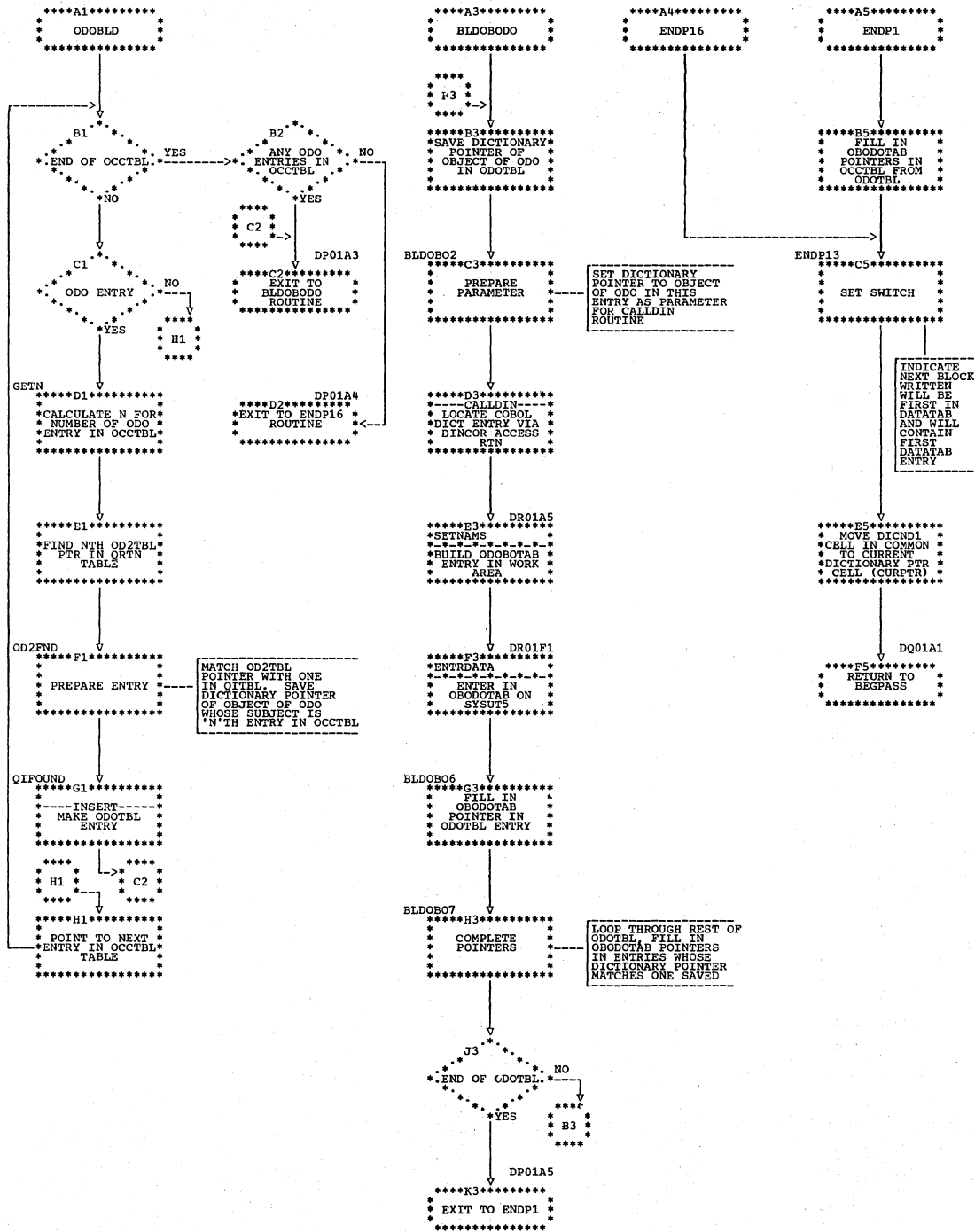






Chart DR. Phase 25: TESTSUBS and SETNAMS Routines

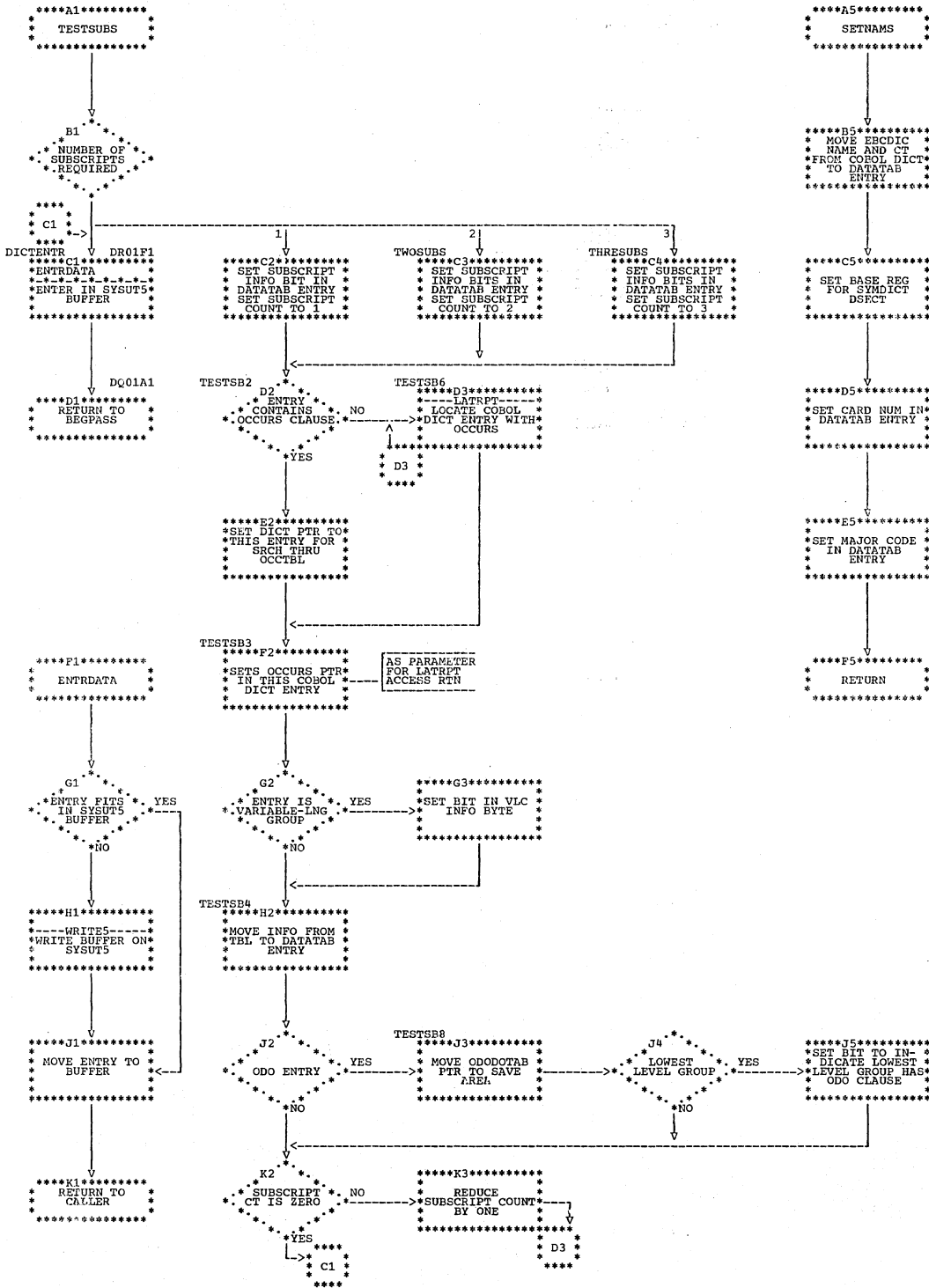


Chart EA. Phase 3: Overall Flow

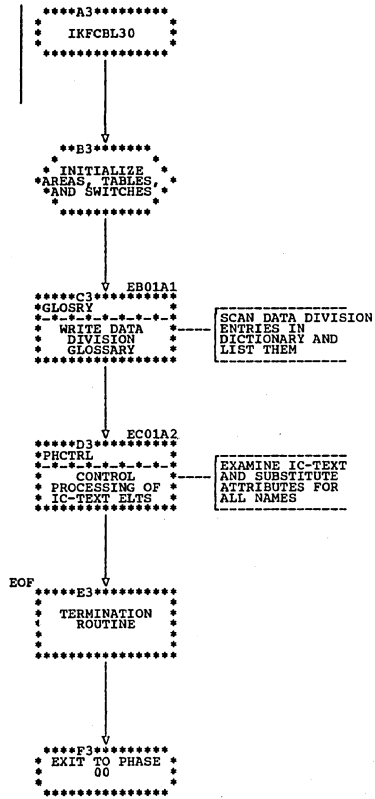


Chart EB. Phase 3: GLOSRY Routine

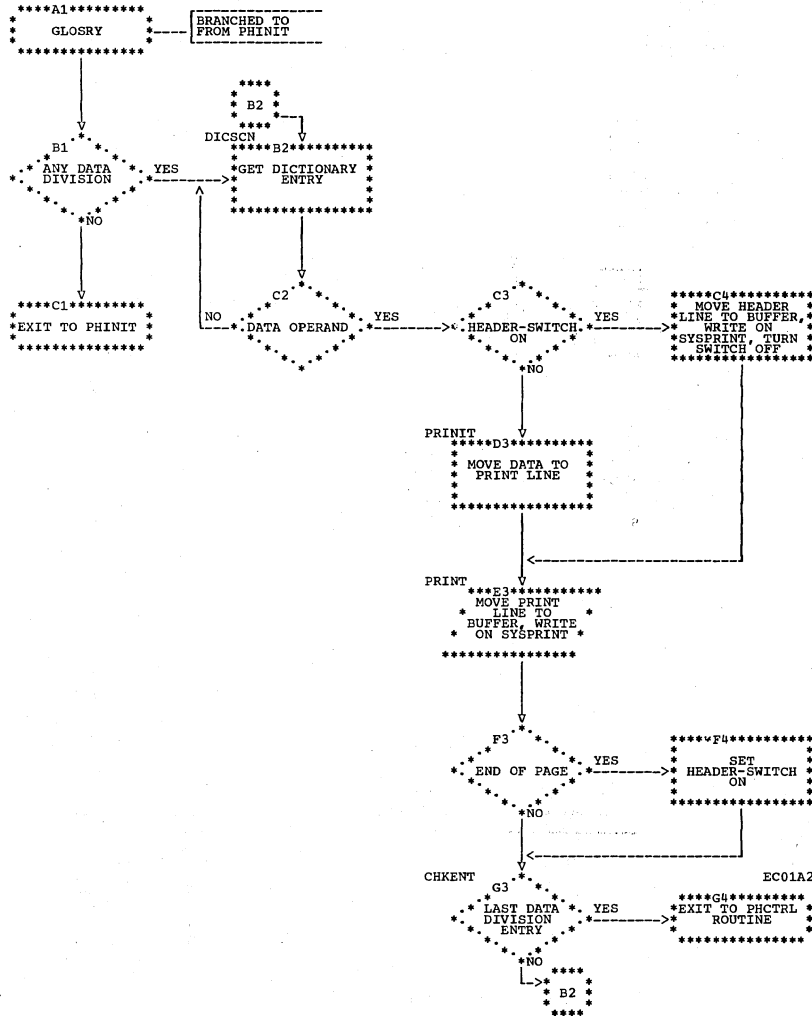




Chart ED (Part 1 of 5). Phase 35: PHCTRL Main Control Routine

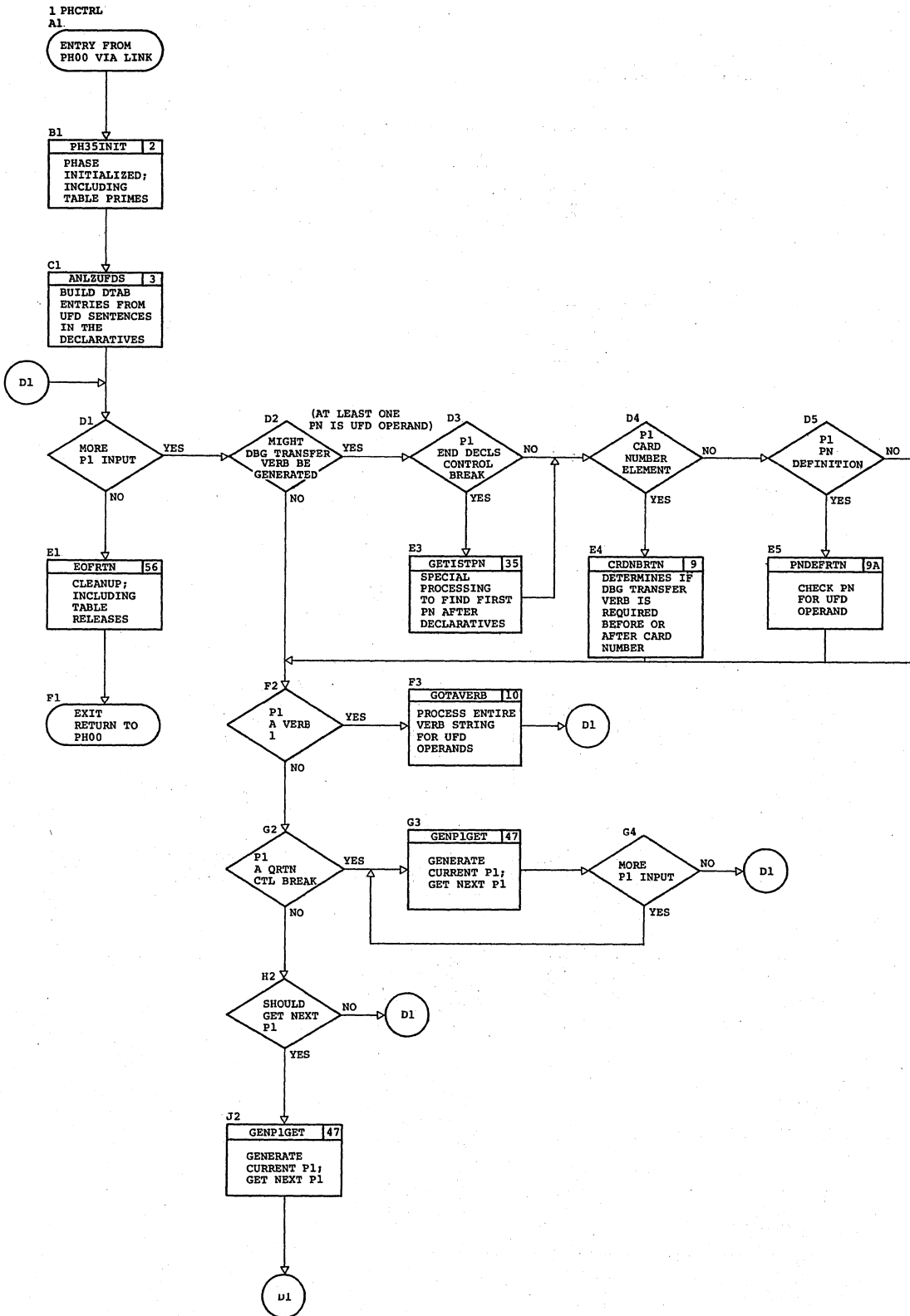


Chart ED (Part 2 of 5). Phase 35: ANLZUFDS Routine

3  
ANLZUFDS  
A1

Page 2 of 5

Chart ED. Phase 35: ANLZUFDS Routine

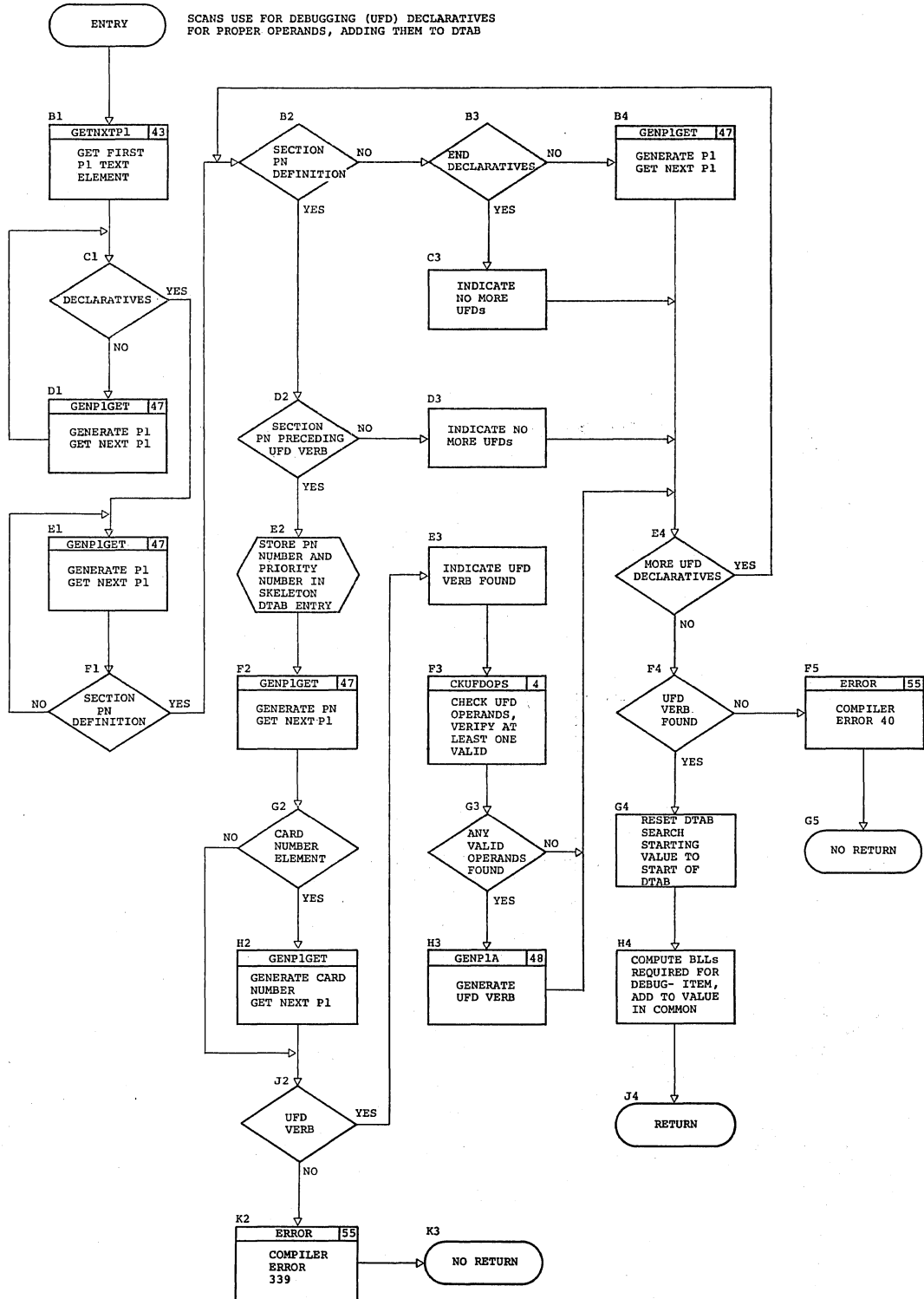
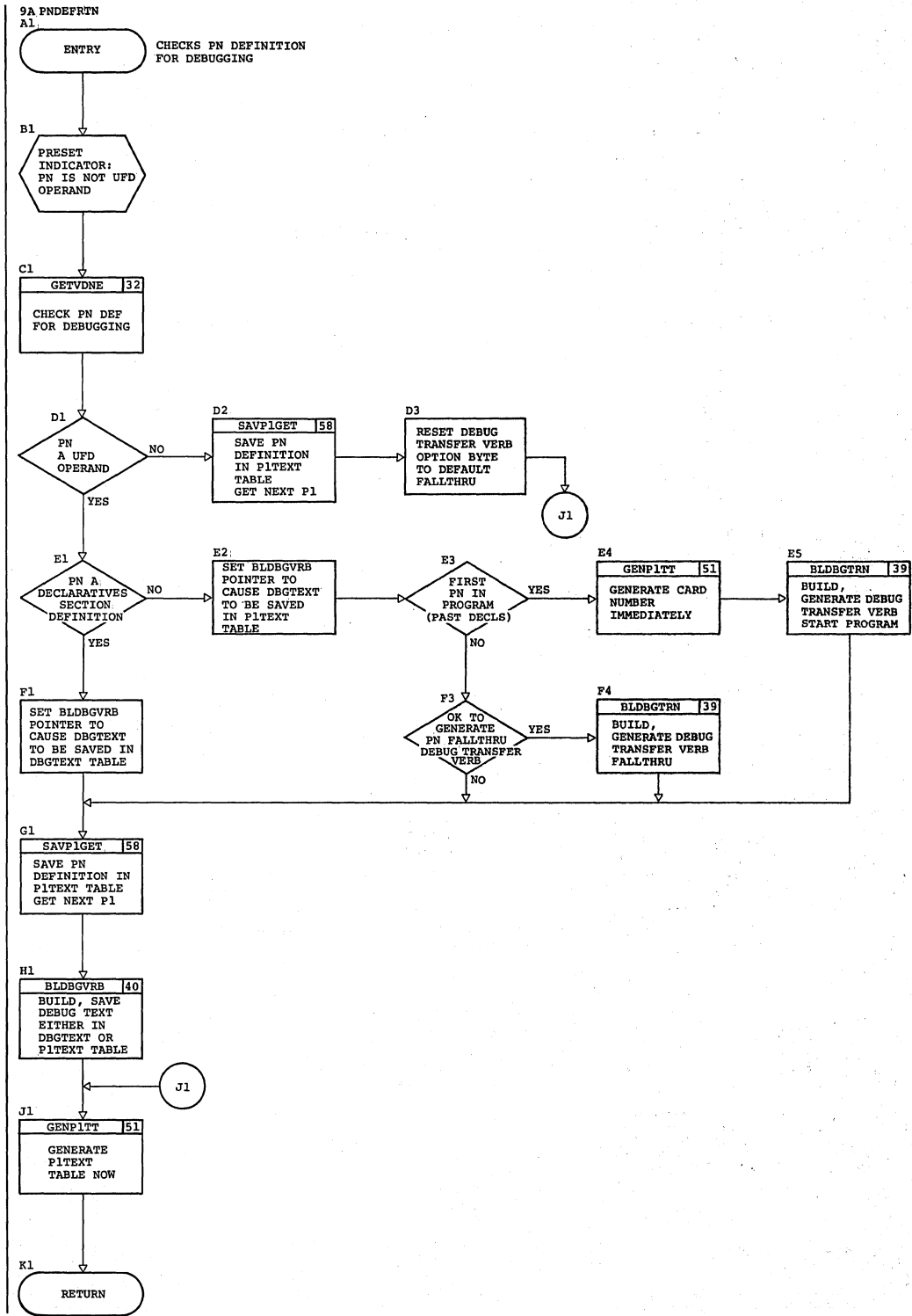


Chart ED (Part 3 of 5). Phase 35: PNDEFRTN Routine





| Chart ED (Part 4 of 5). Phase 35: GOTAVERRB Routine

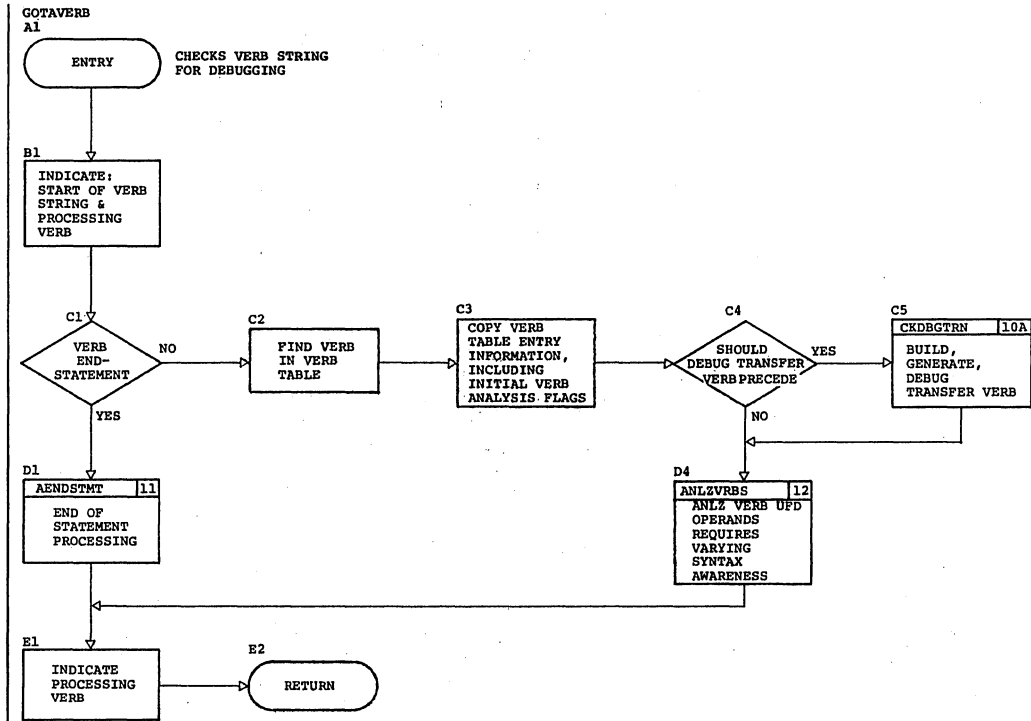


Chart ED (Part 5 of 5). Phase 35: ANLZVRBS Routine

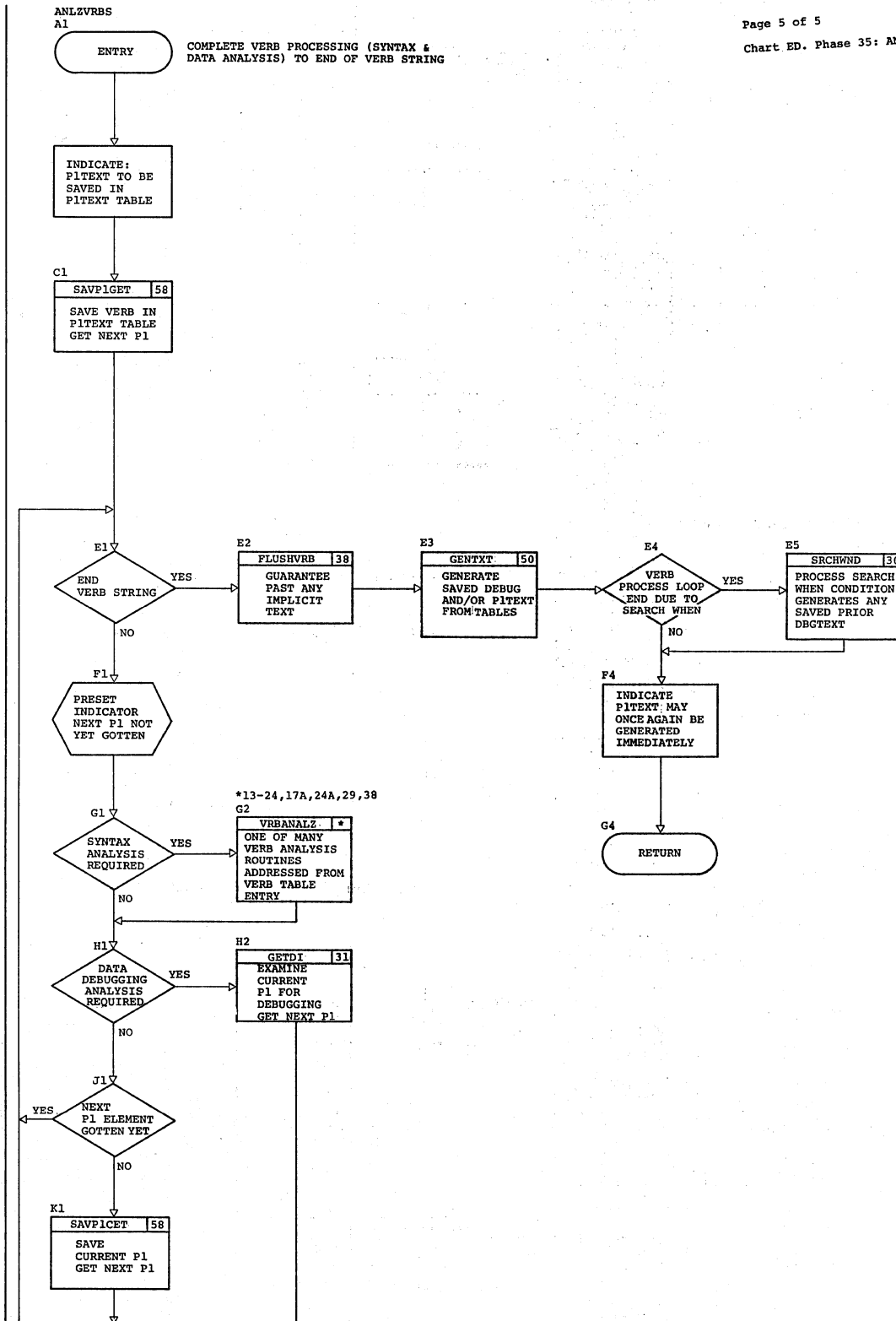


Chart FA. Phase 4: Overall Flow

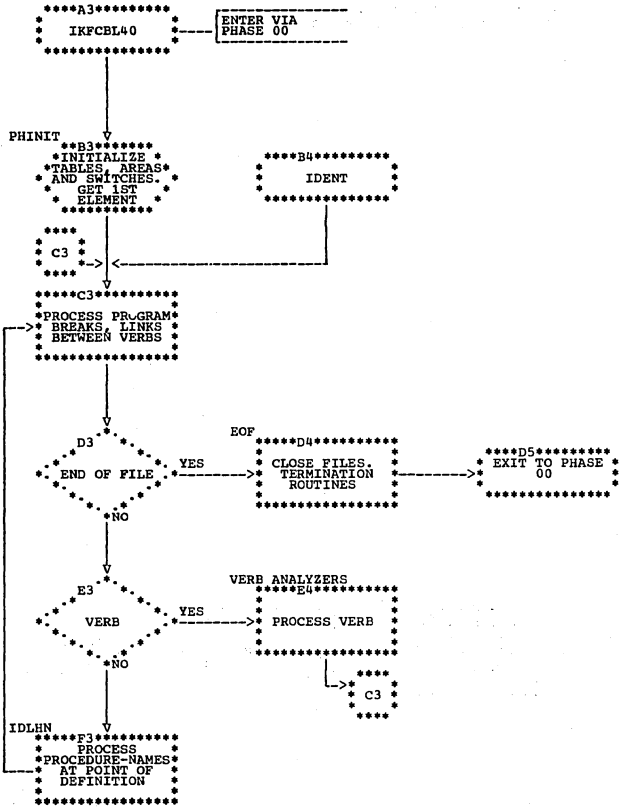


Chart FB. Phase 4: IF Routine

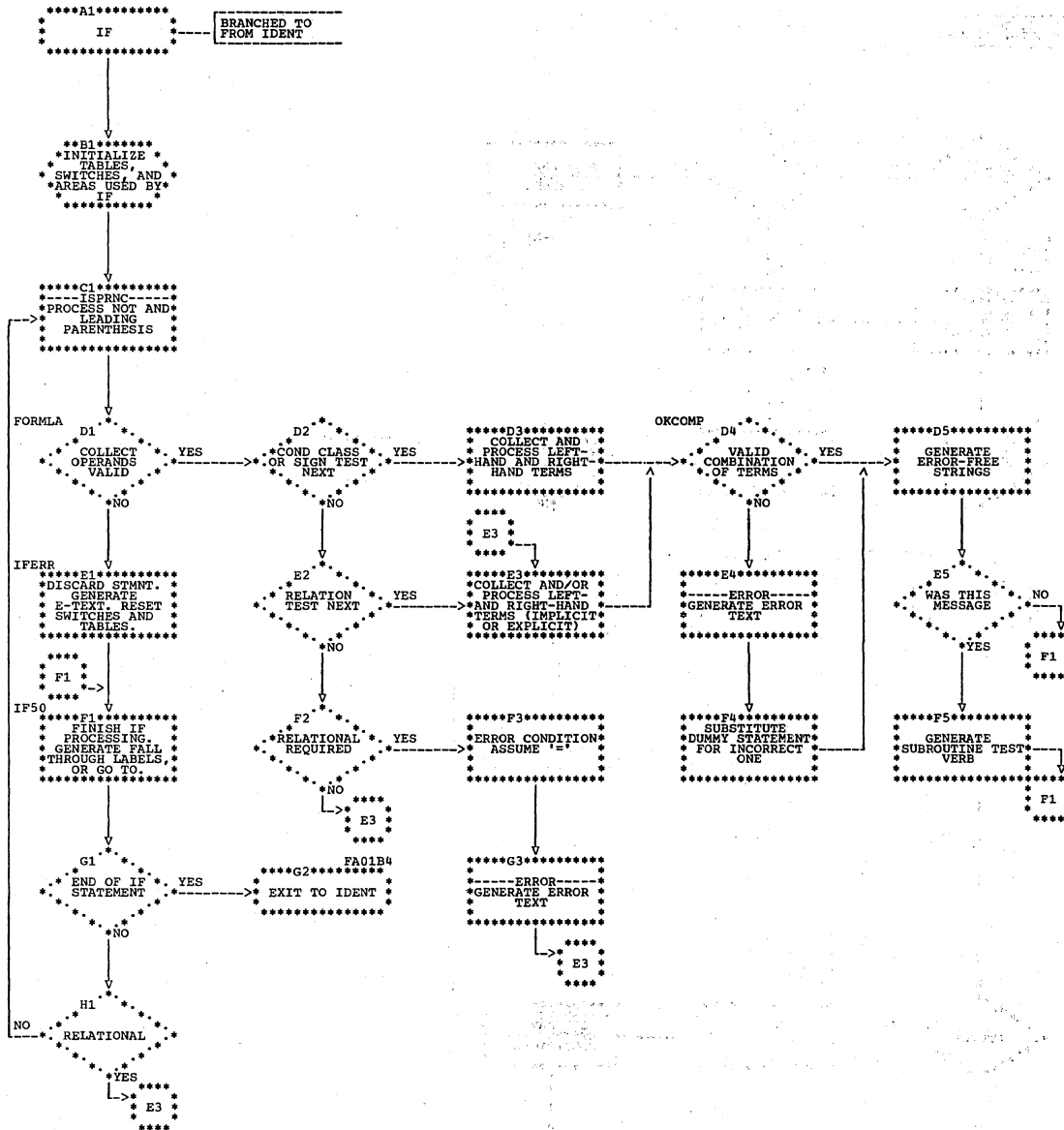


Chart FC. Phase 4: PRFORM Routine

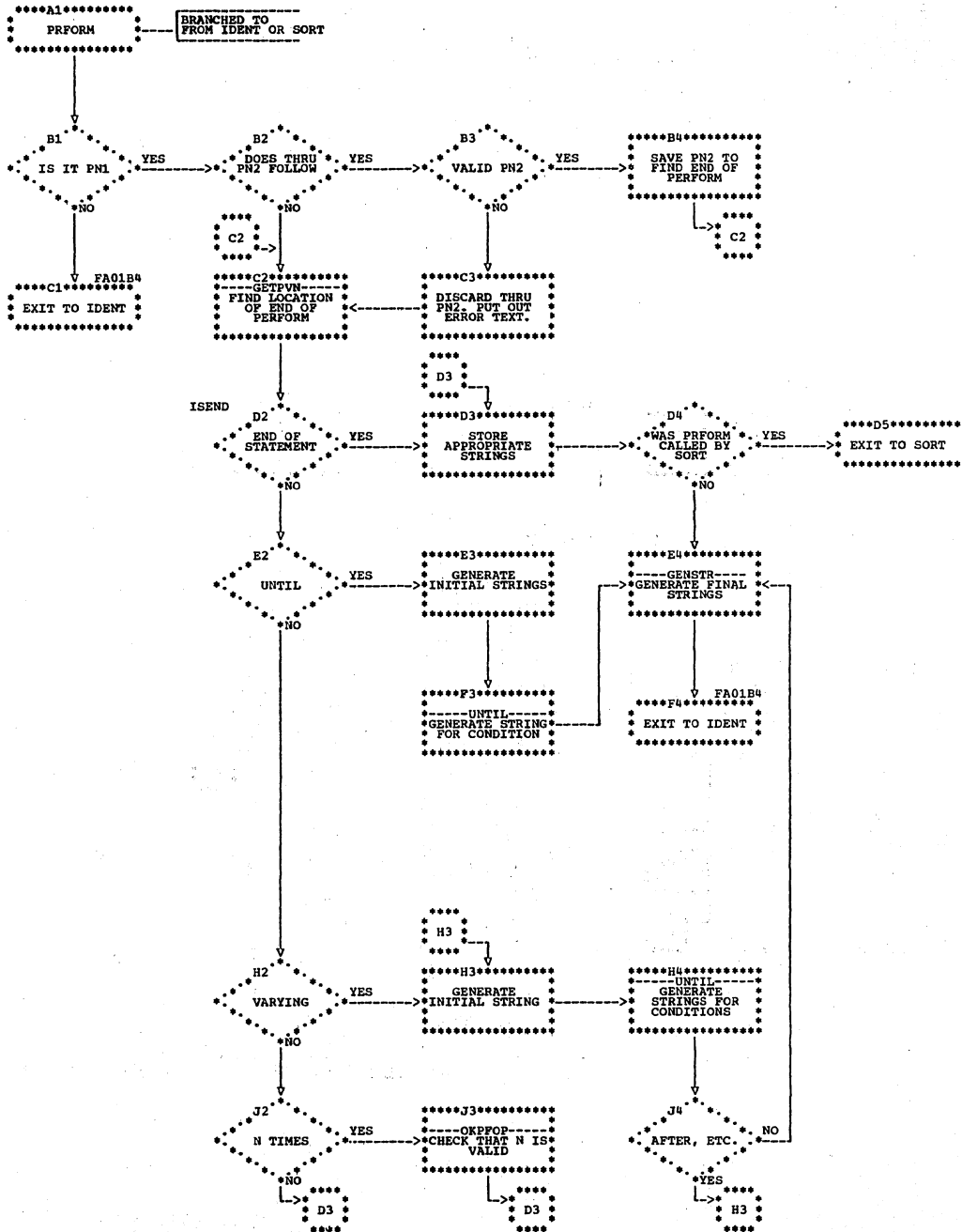


Chart FD. Phase 45: Overall Flow

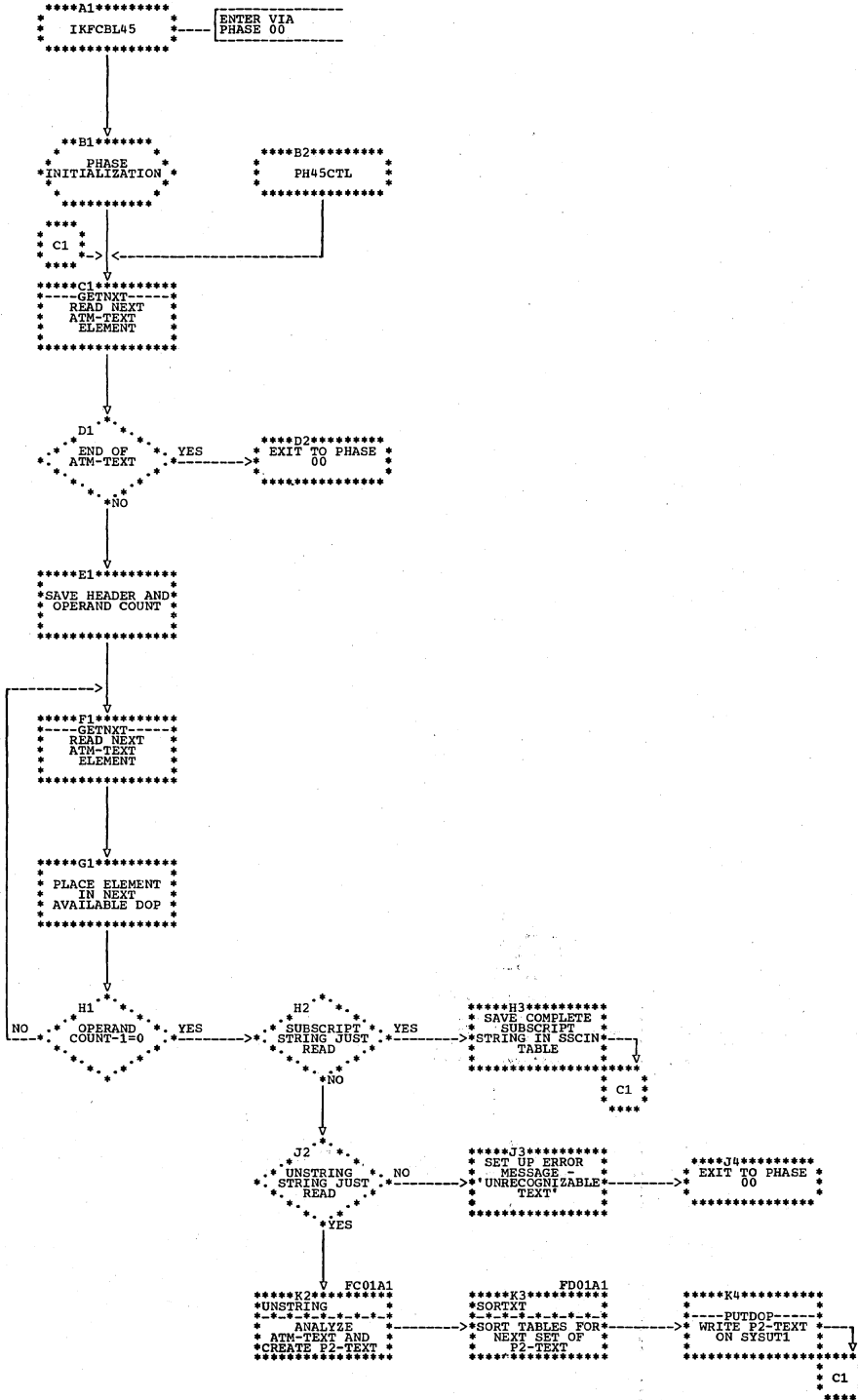


Chart FE. Phase 45: UNSTRING Routine

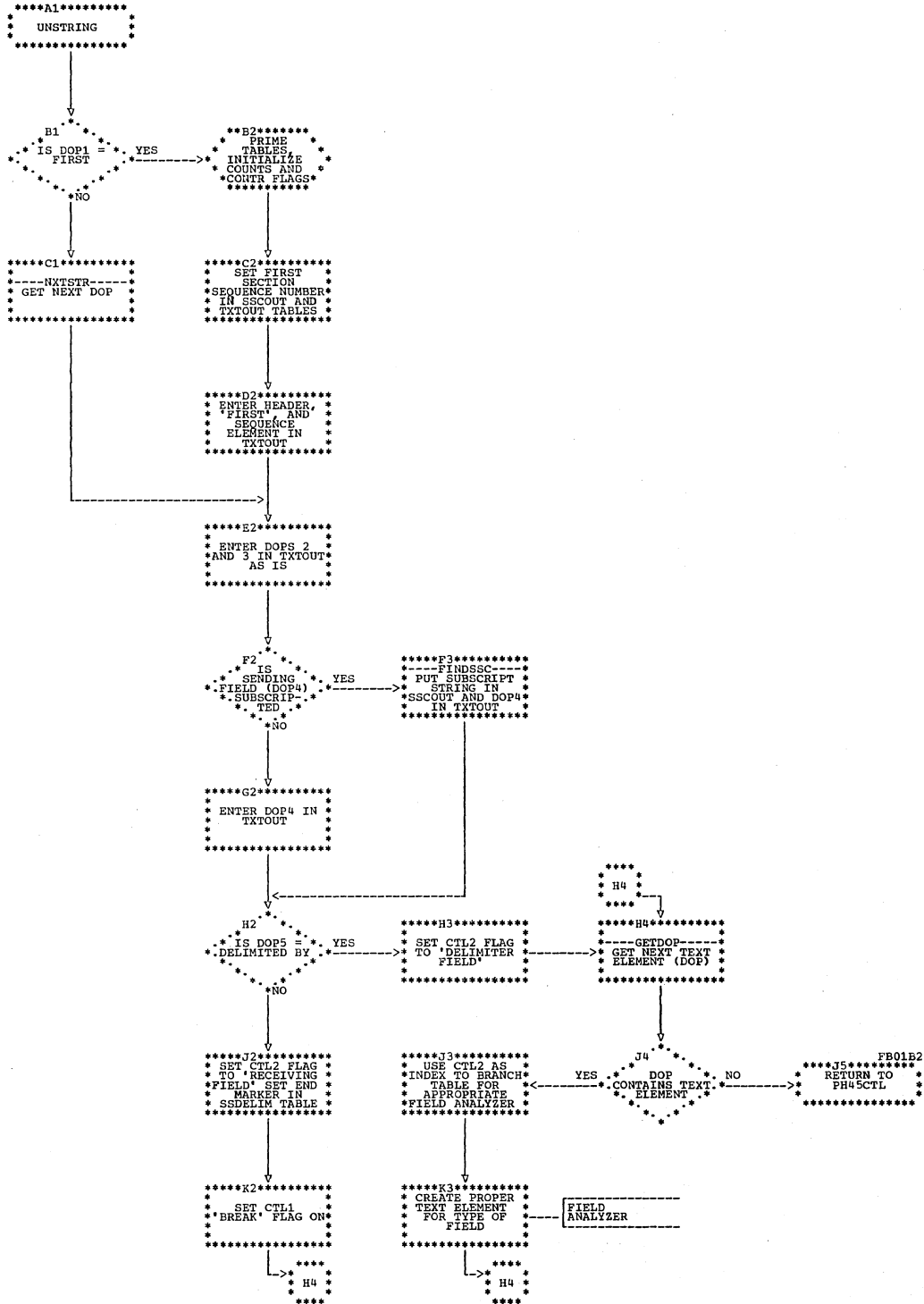


Chart FF. Phase 45: SORTXT Routine

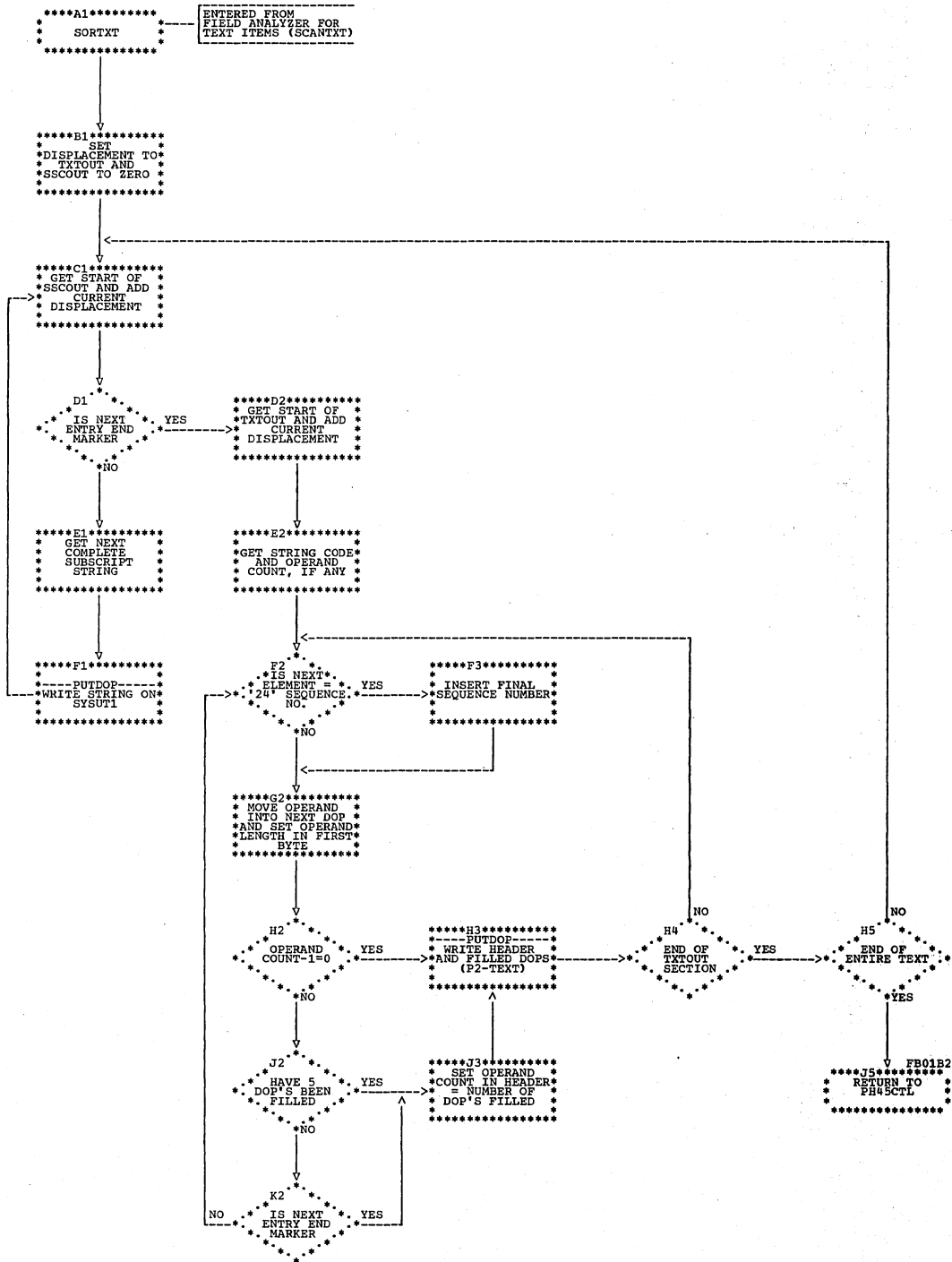




Chart GA. Phase 50: Overall Flow

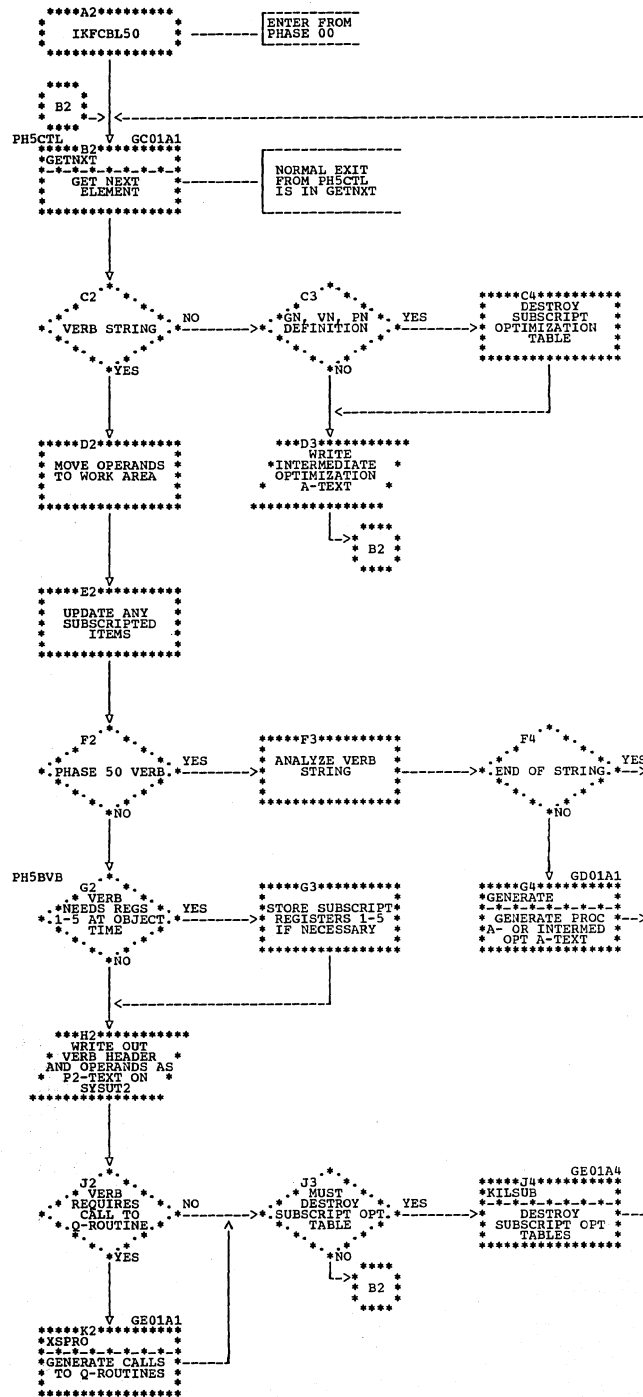


Chart GB. Phase 50: PH5CTL Routine

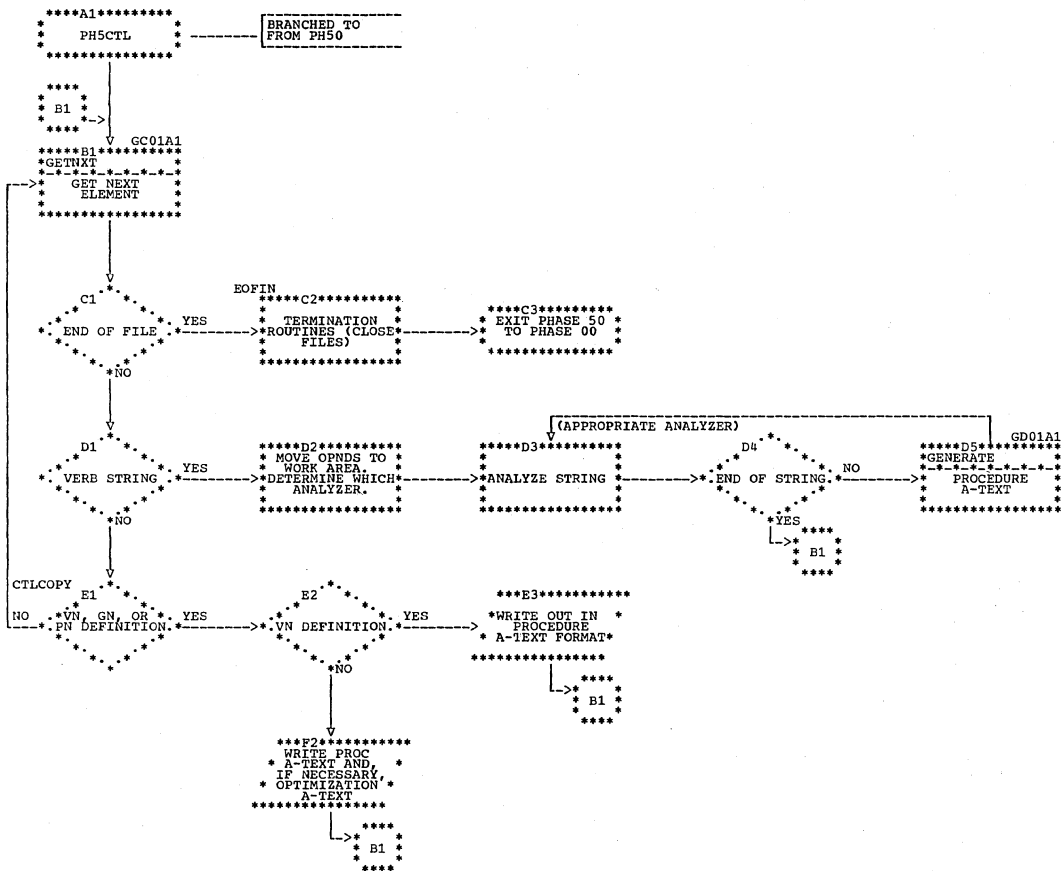


Chart GC (Part 1 of 2). Phase 50: GETNXT Routine

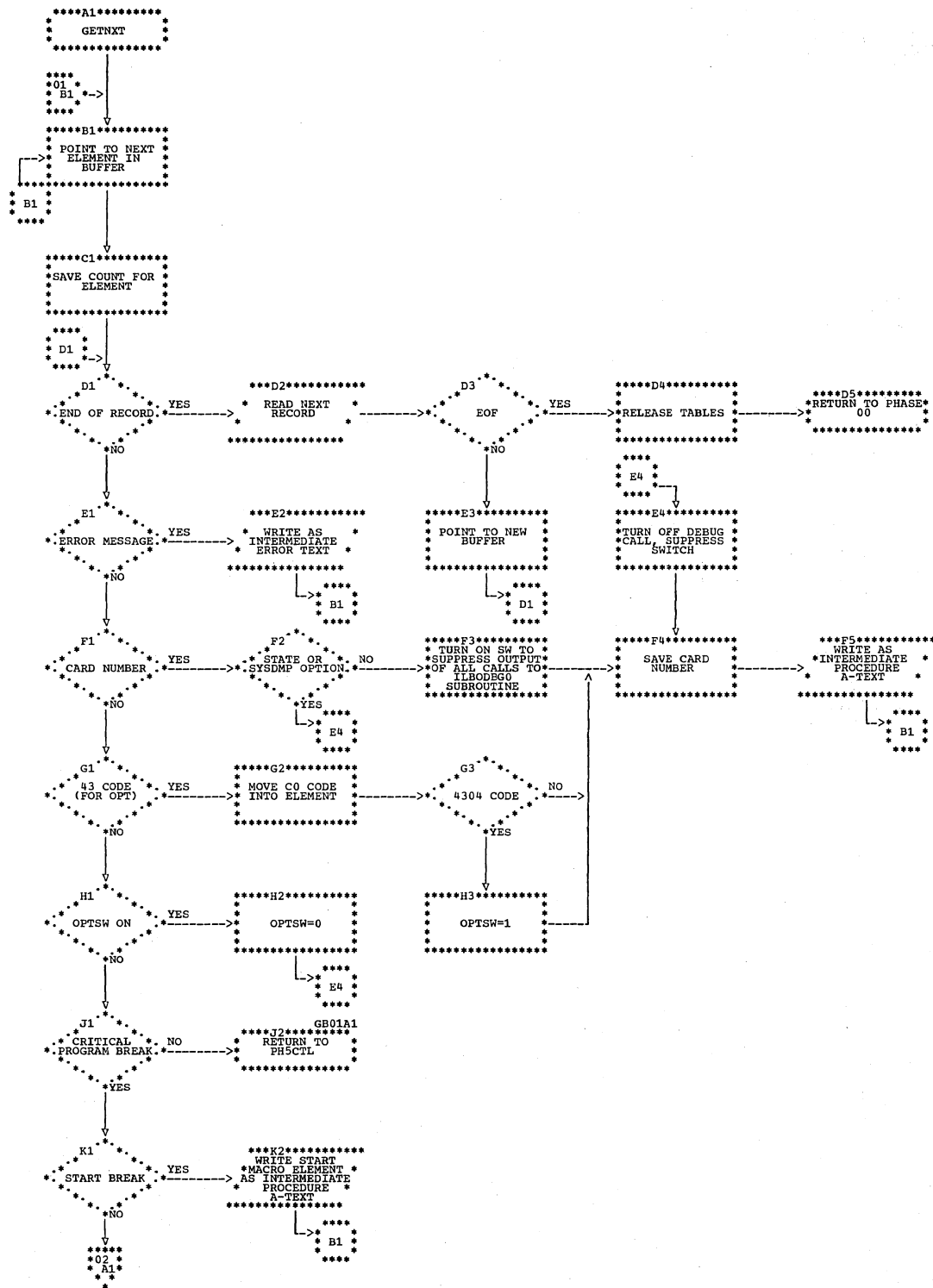


Chart GC (Part 2 of 2). Phase 50: GETNXT Routine

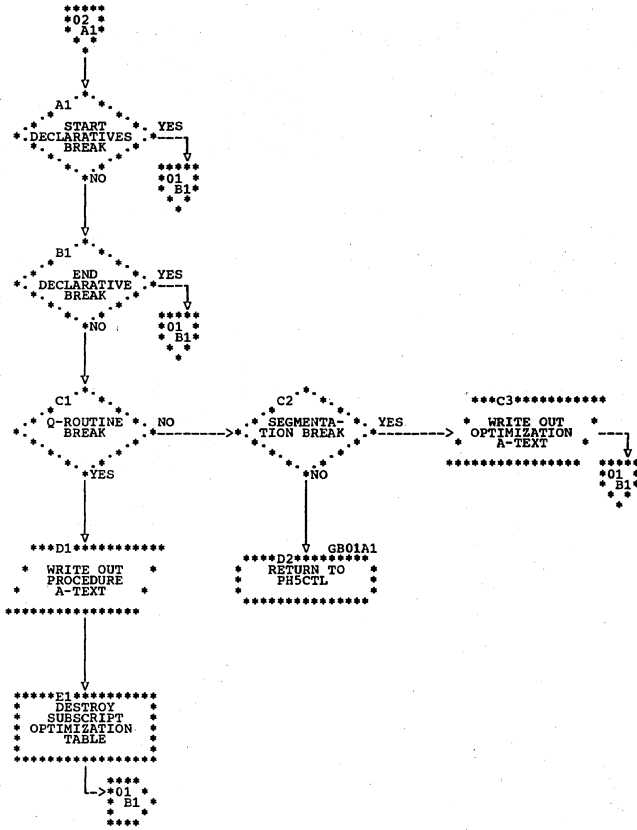


Chart GD. Phase 50: Generate Routine

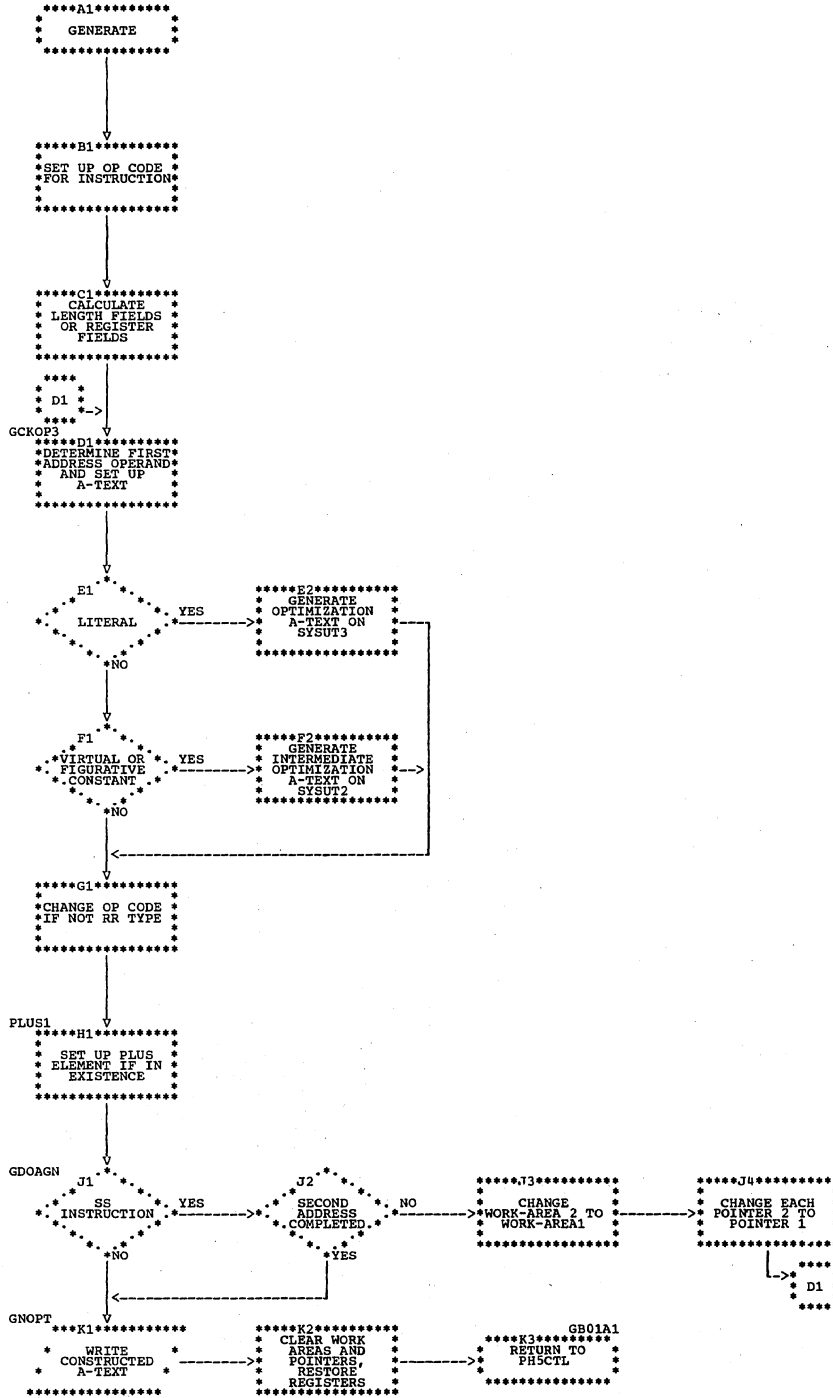


Chart GE. Phase 50: XSPRO and KILSUB Routines

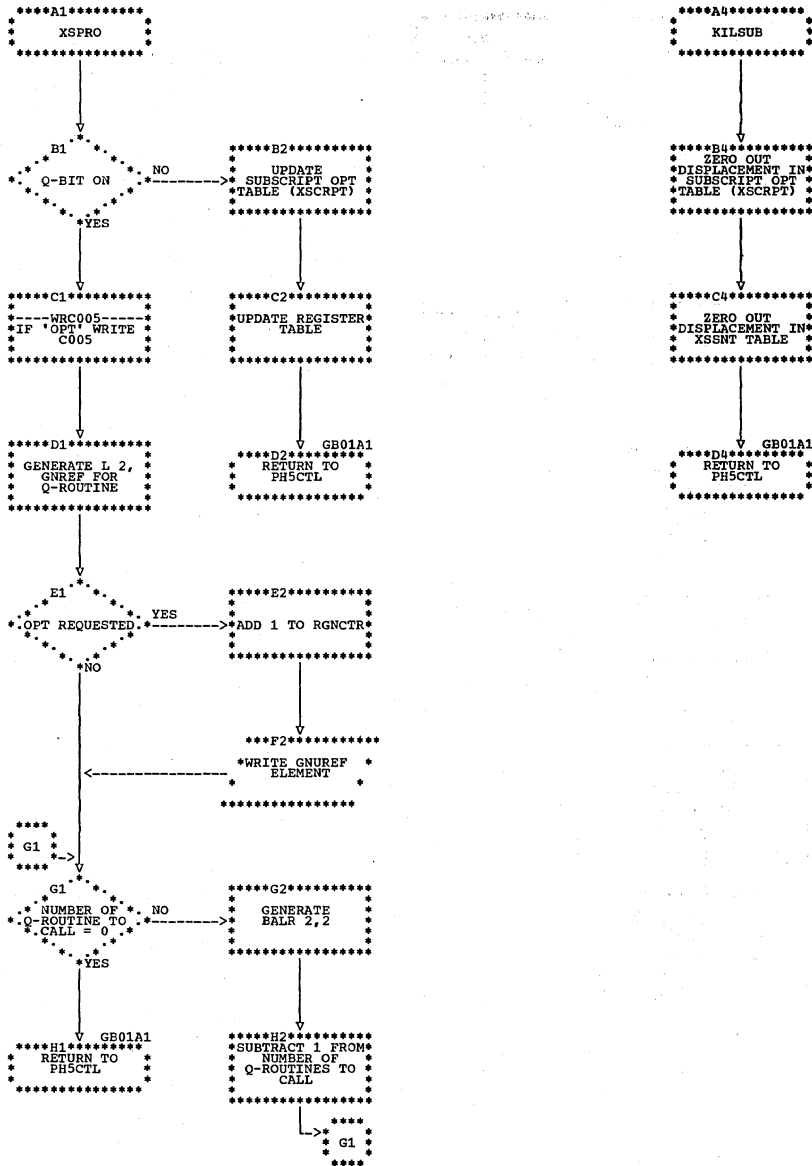


Chart GF. Phase 51: Overall Flow

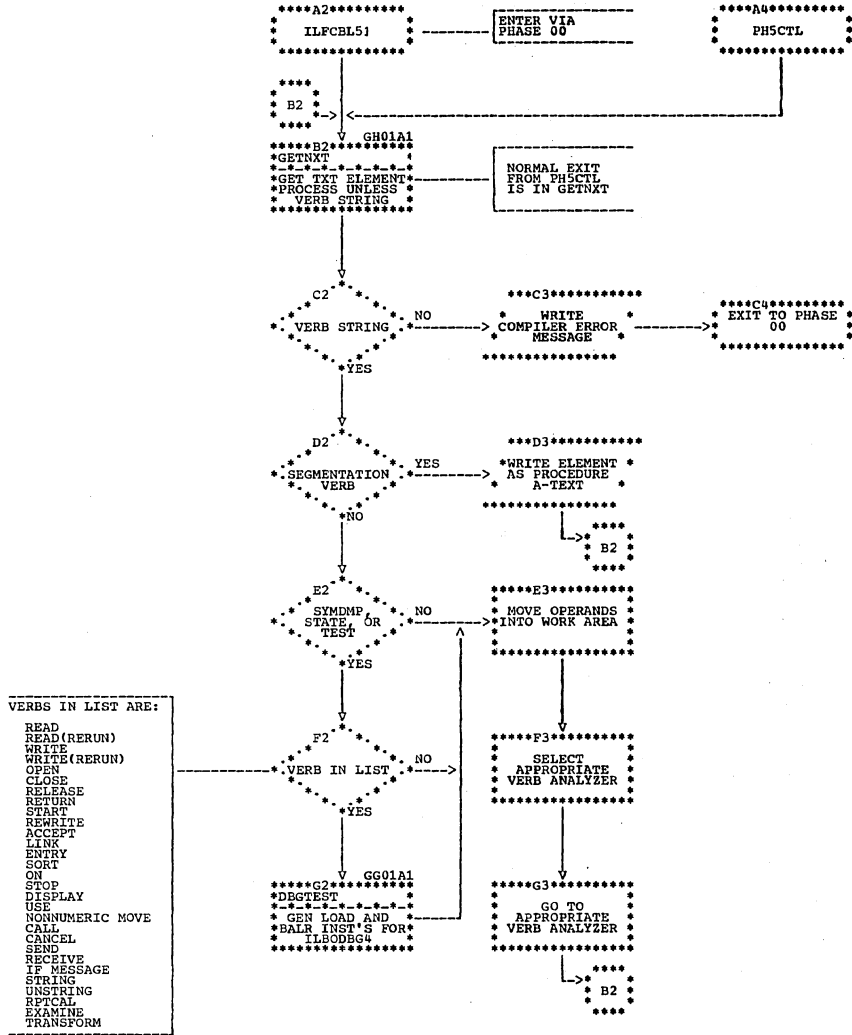


Chart GG. Phase 51: DBGTEST Routine

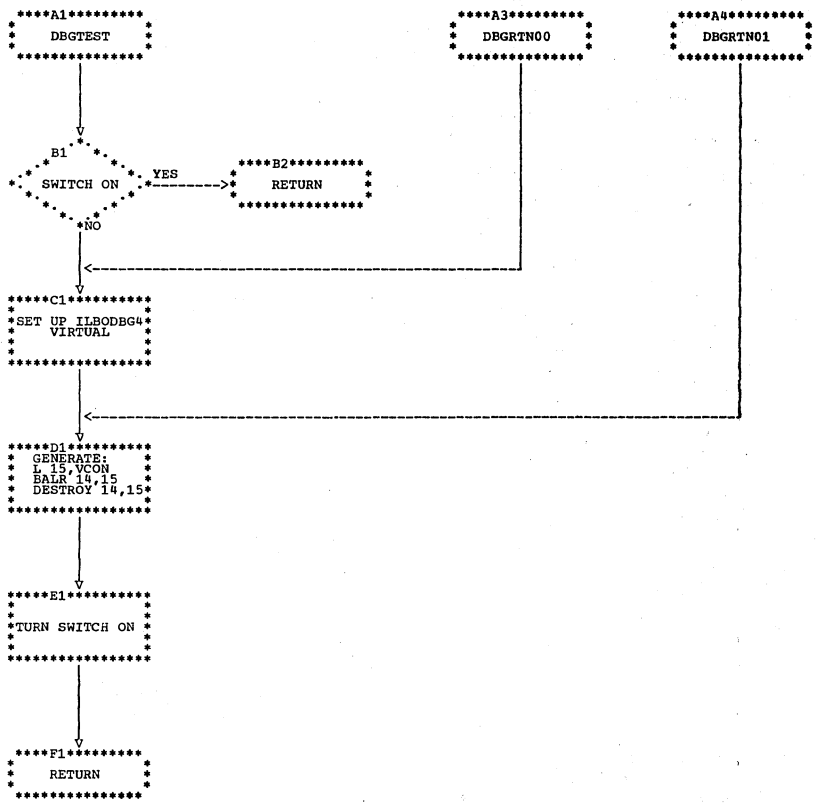




Chart GH. Phase 51: GETNXT Routine

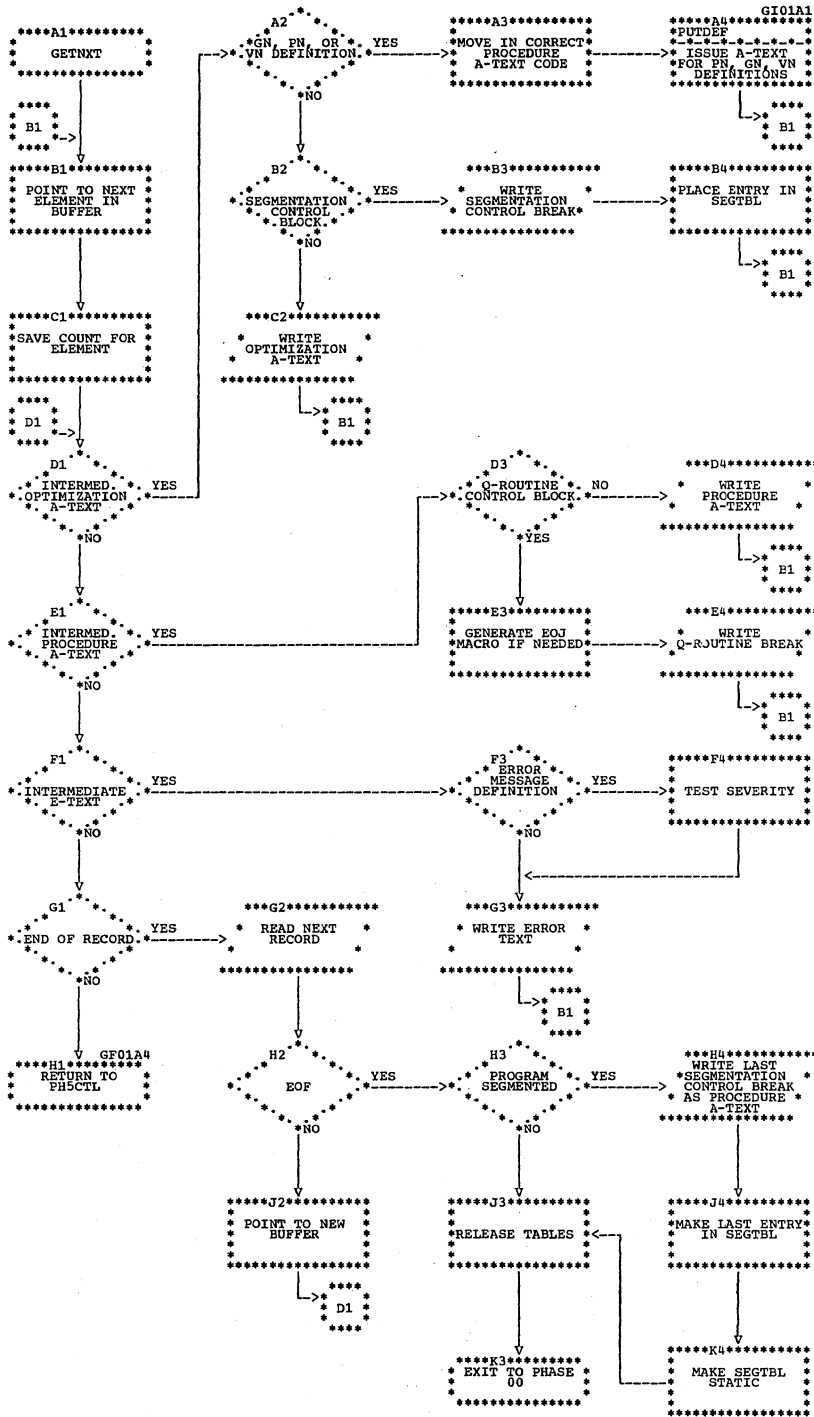


Chart G1. Phase 51: PUTDEF Routine

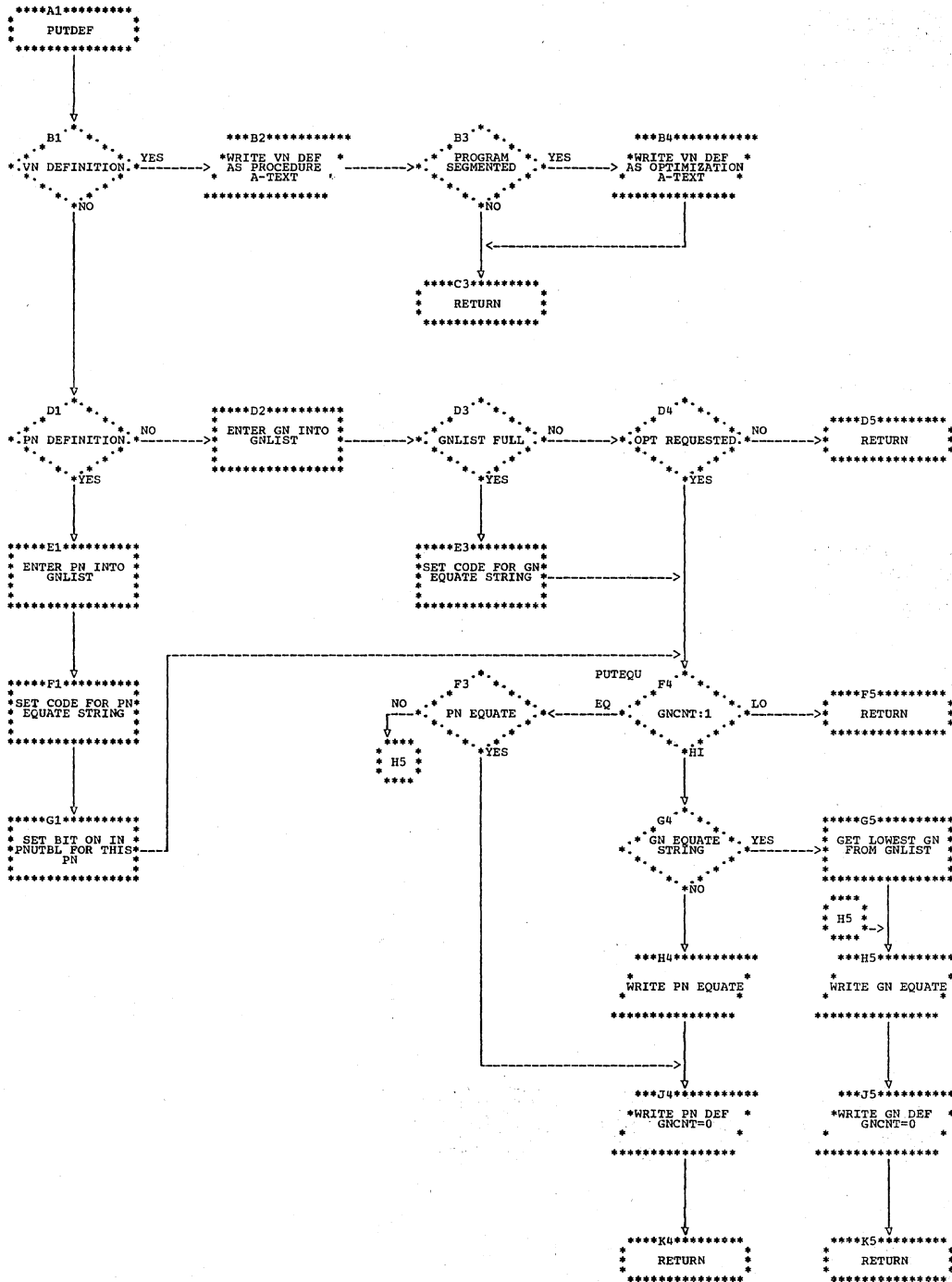


Chart GJ. Phase 51: A-text Generator Routine

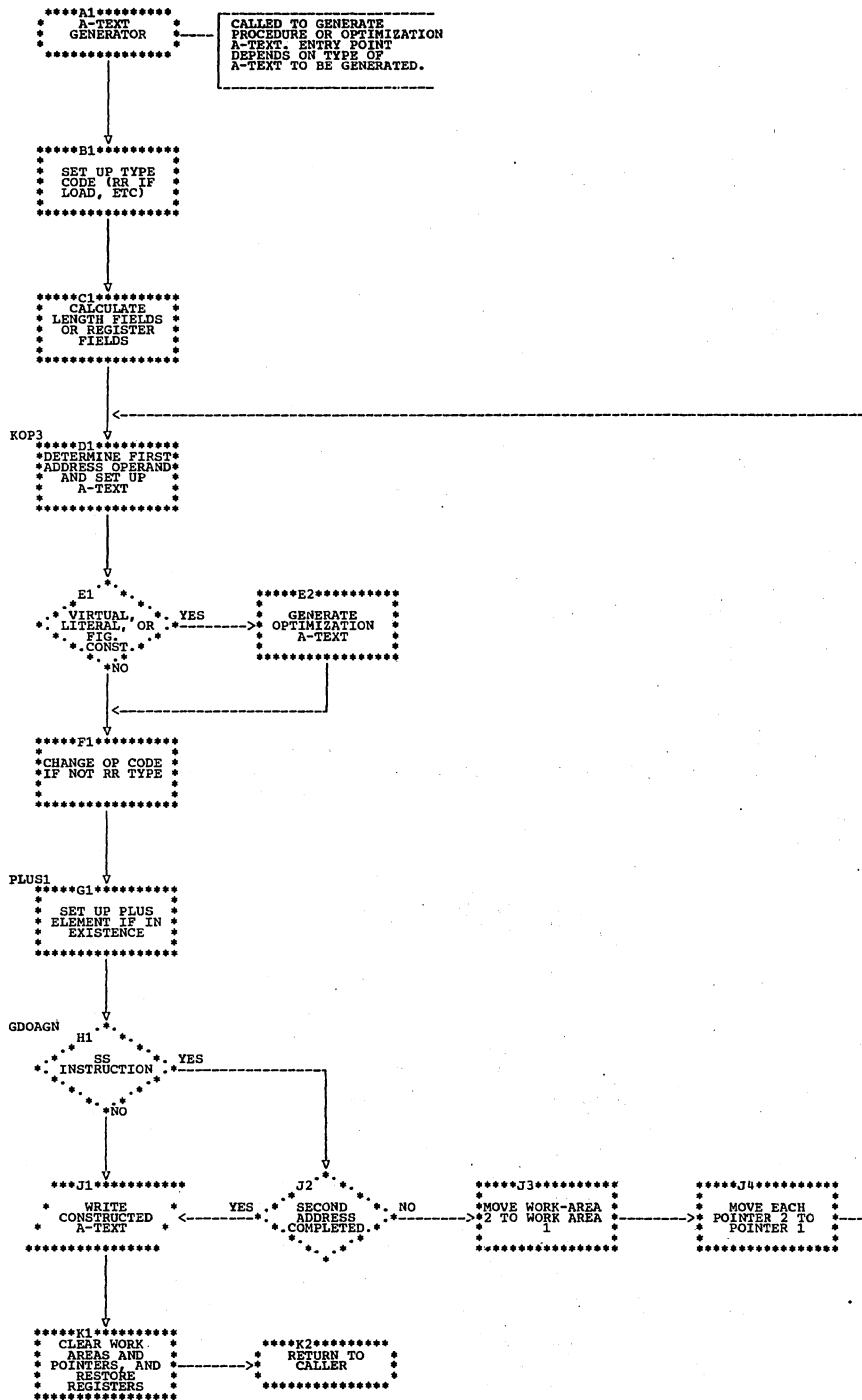


Chart HA. Phase 6: Overall Flow

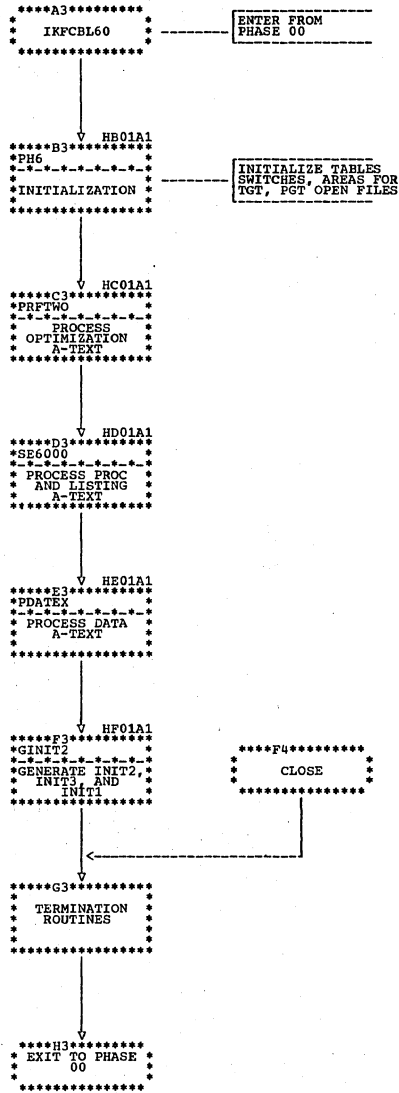


Chart HB. Phase 6: PH6 Routine

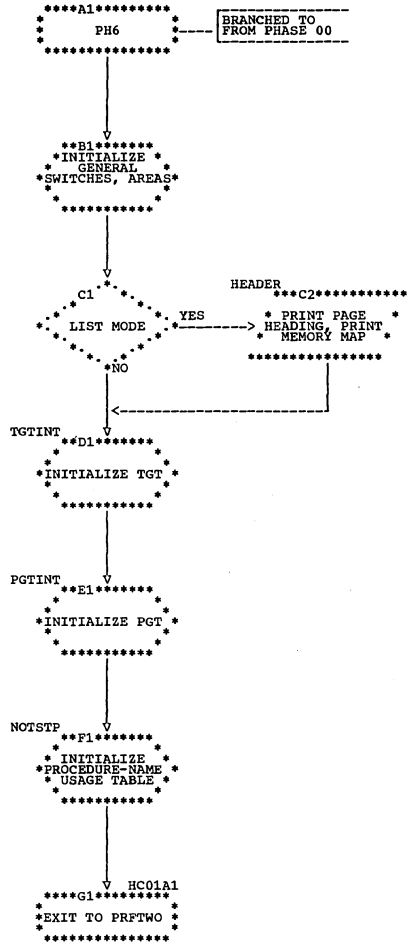


Chart HC. Phase 6: PRFTWO Routine

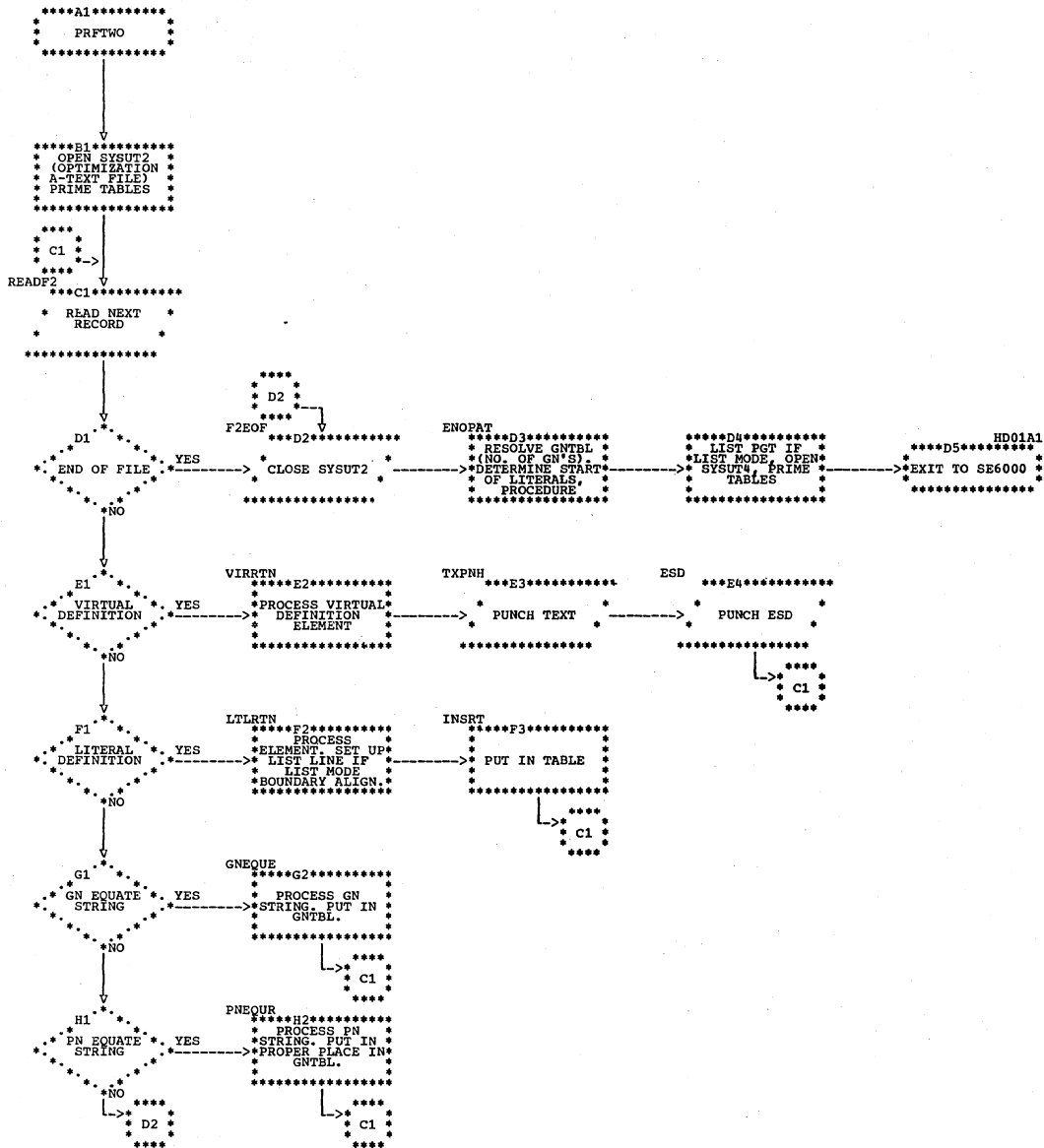


Chart HD. Phase 6: SE6000 Routine

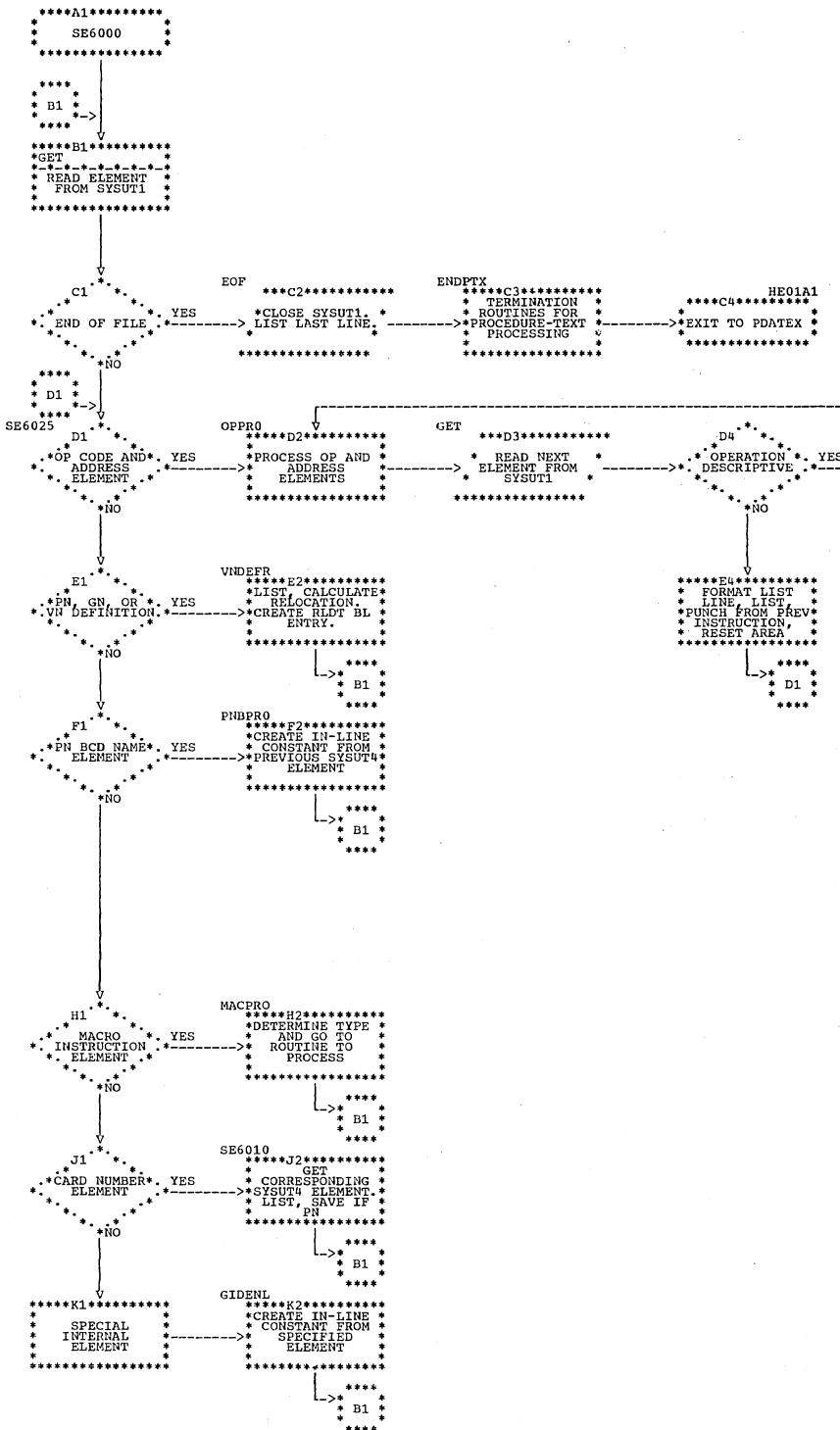


Chart HE. Phase 6: PDATEX Routine

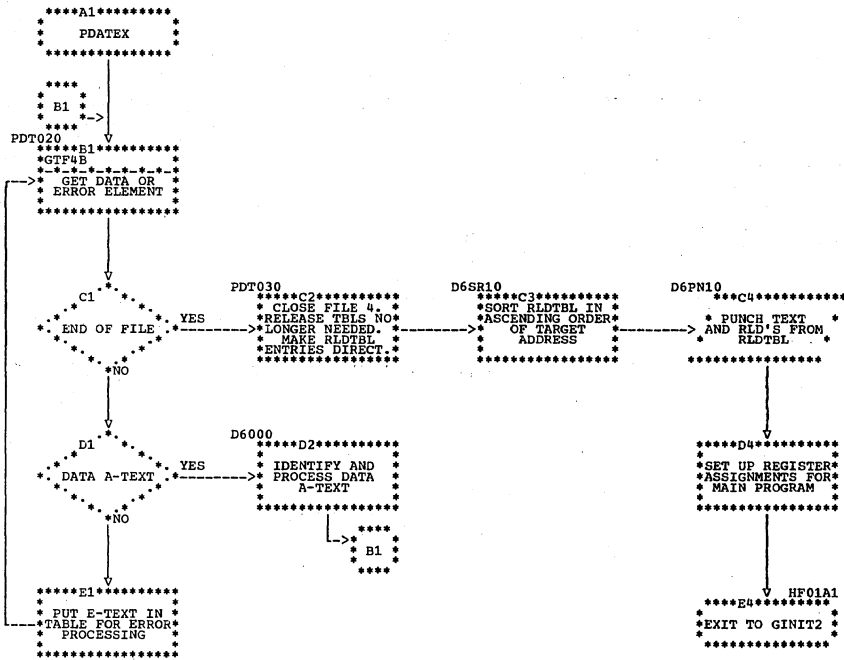




Chart HF. Phase 6: GINIT2 Routine

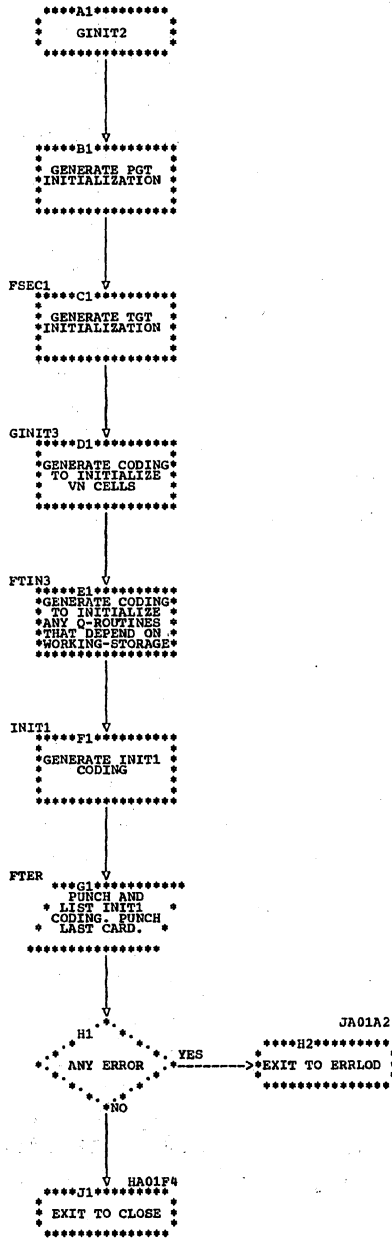


Chart IA. Phase 62: Overall Flow

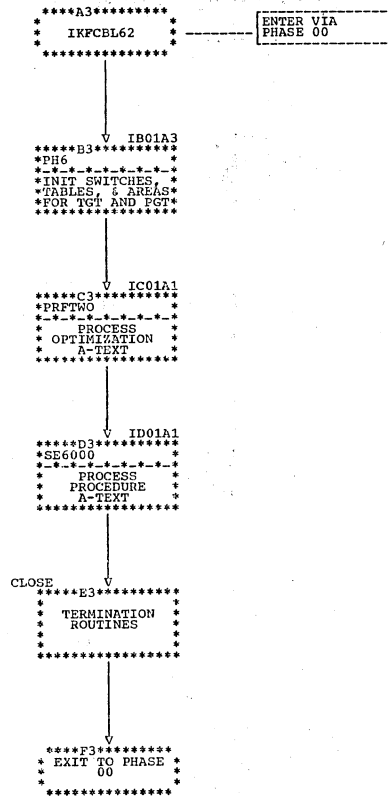


Chart IB. Phase 62: PH6 Routine

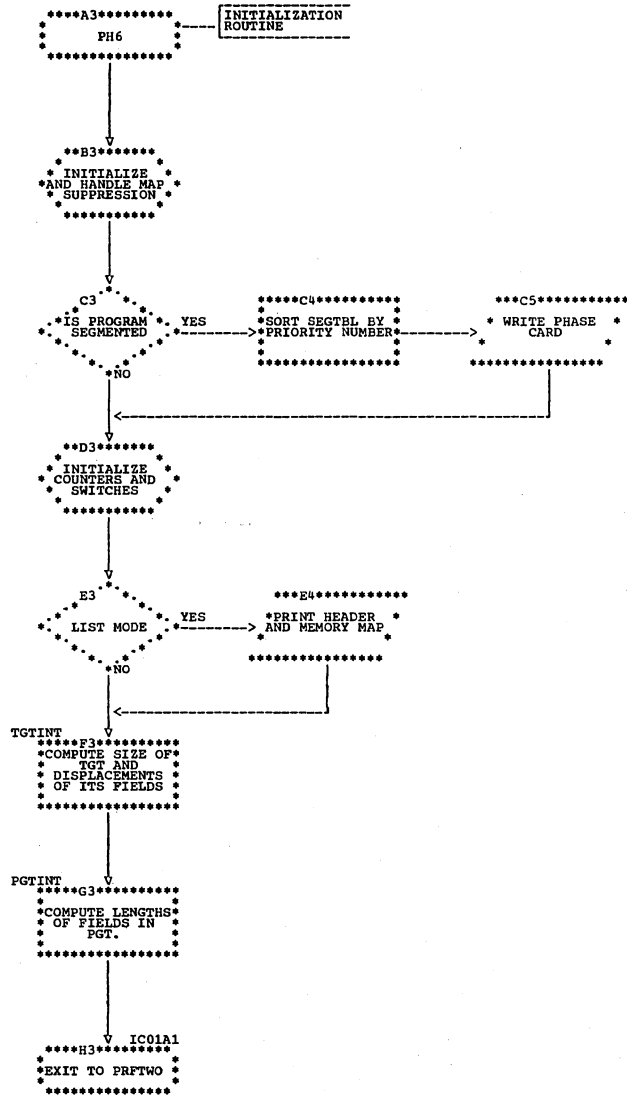


Chart IC. Phase 62: PRFTWO Routine

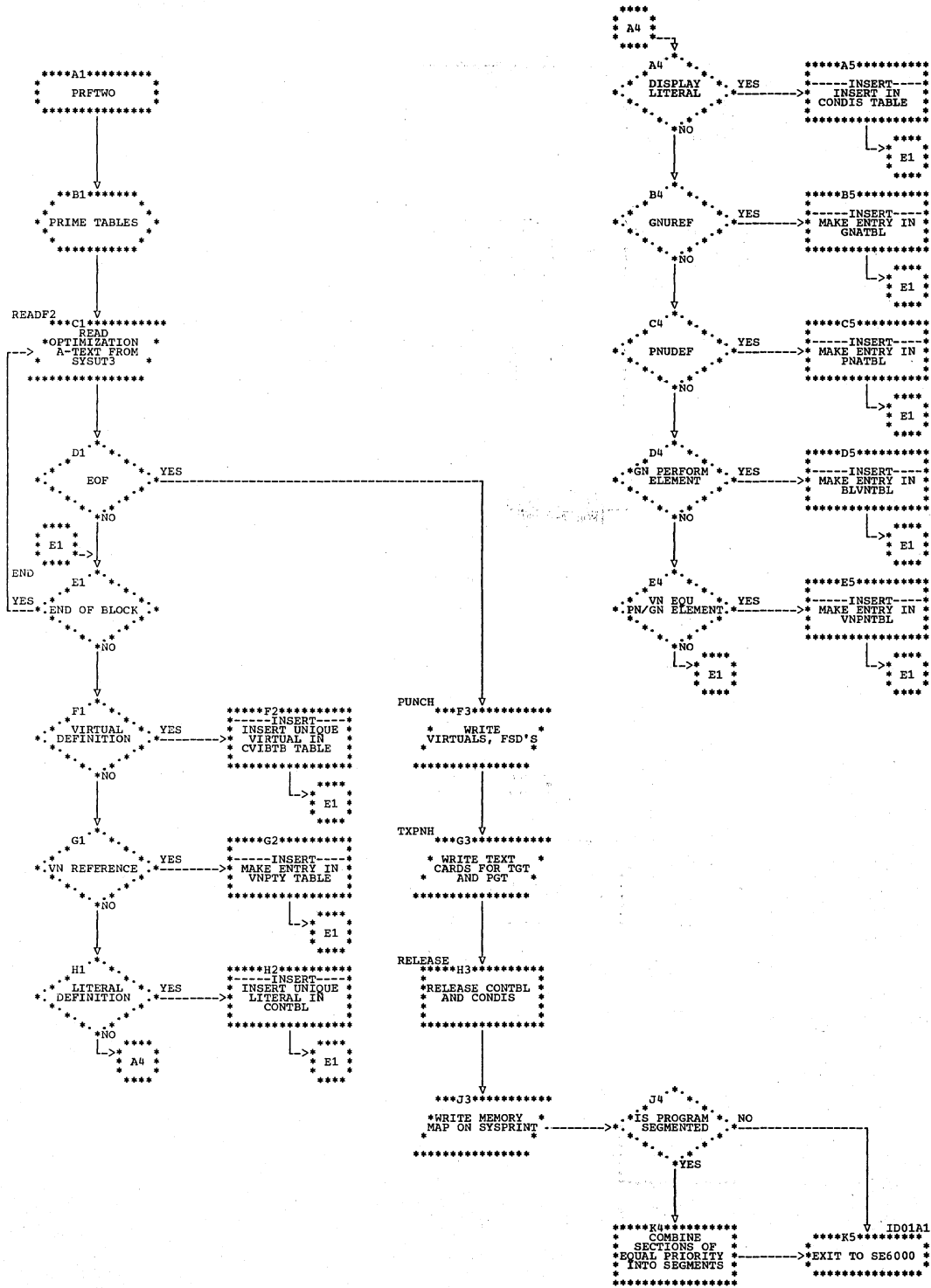


Chart ID (Part 1 of 2). Phase 62: SE6000 Routine

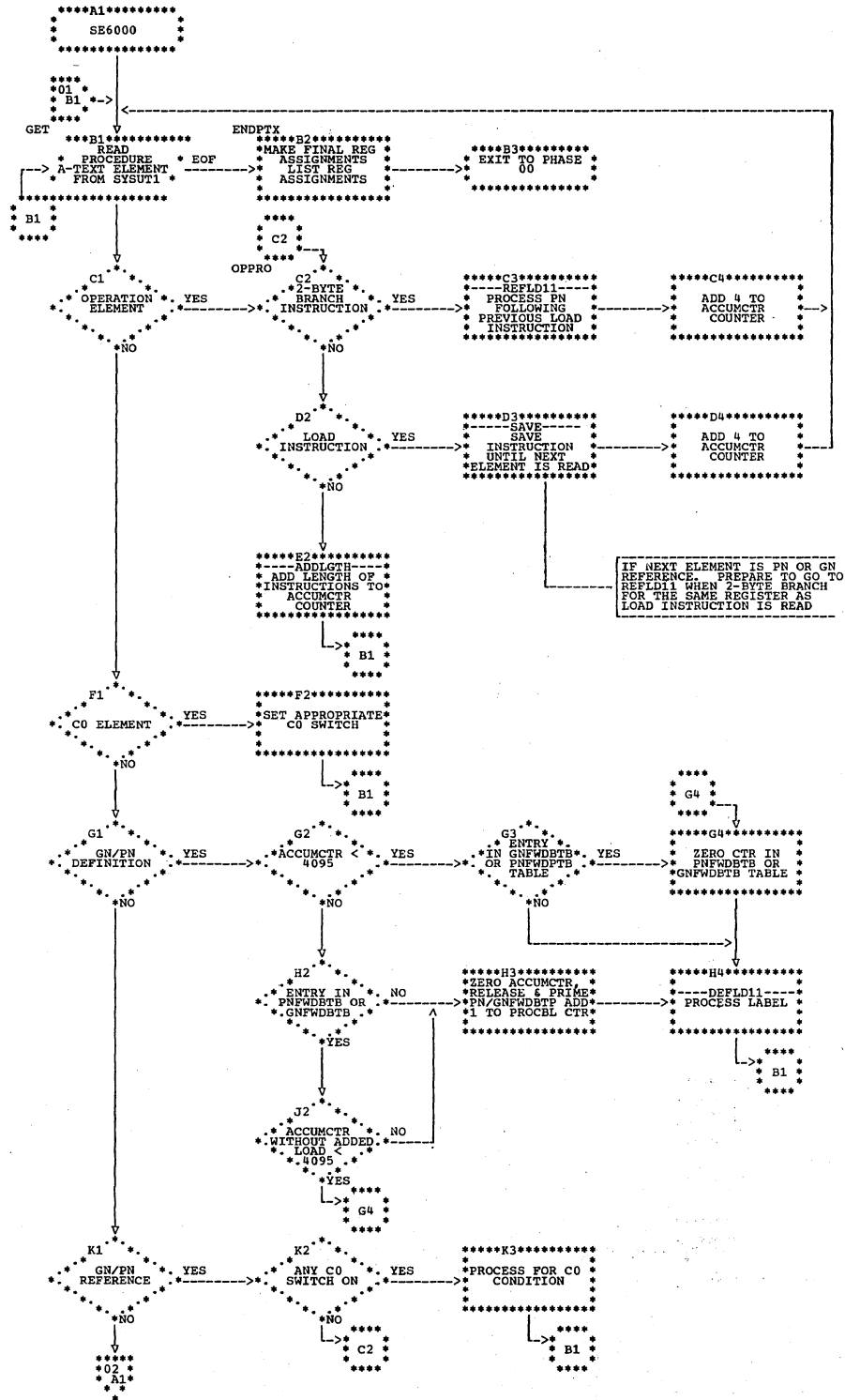


Chart ID (Part 2 of 2). Phase 62: SE6000 Routine

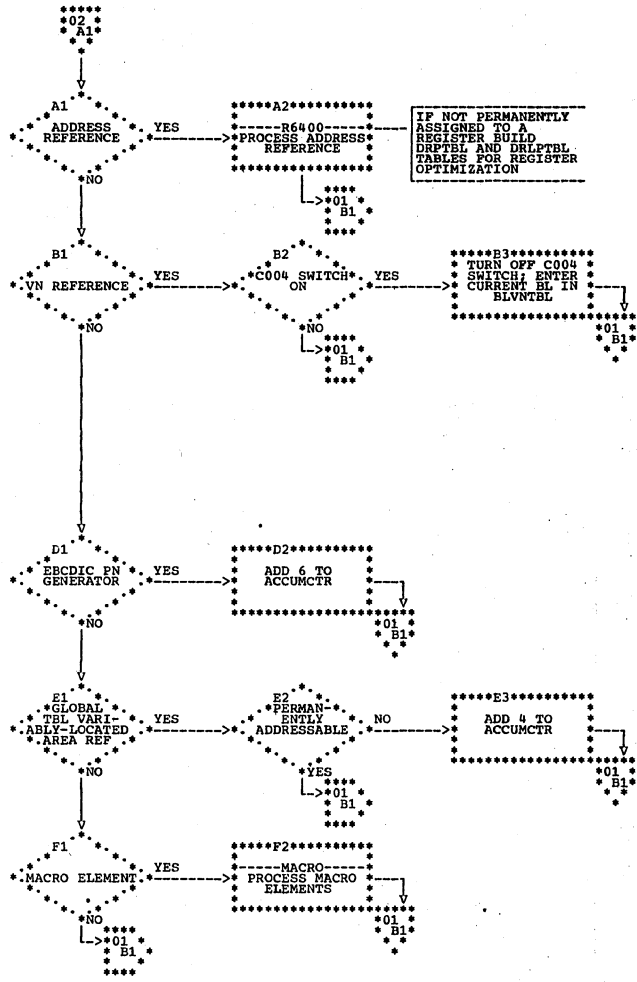


Chart IE. Phase 63: Overall Flow

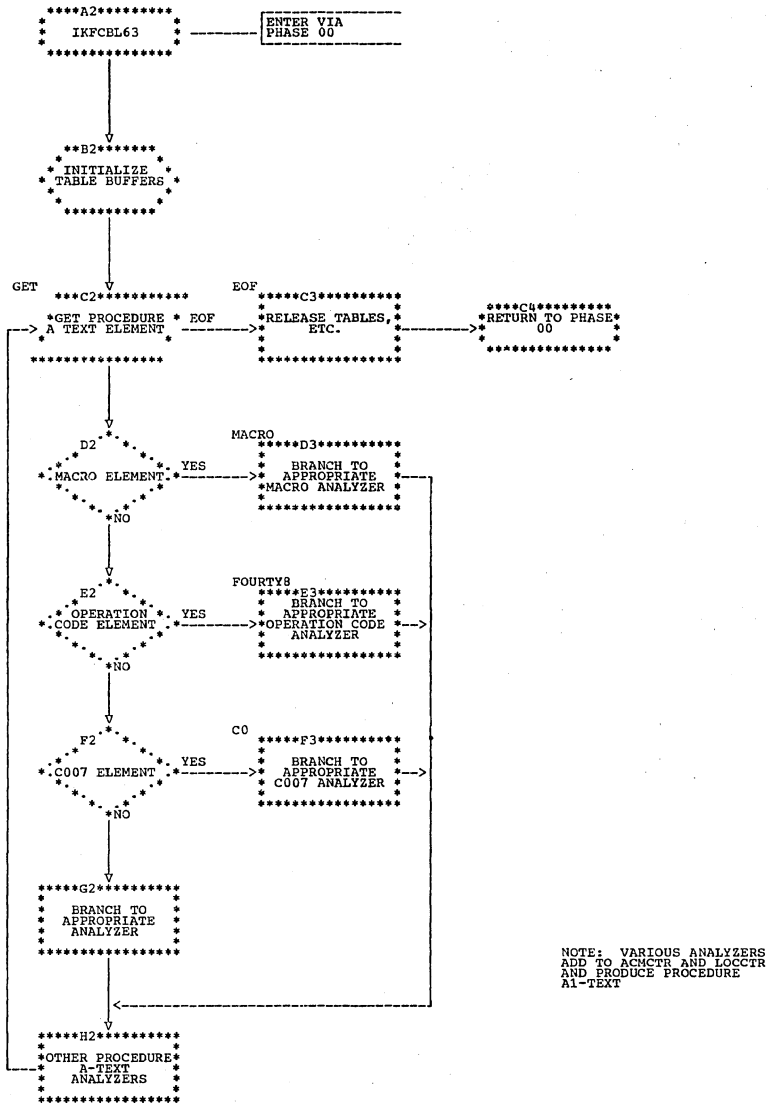


Chart IF. Phase 63: BRANCH Routine

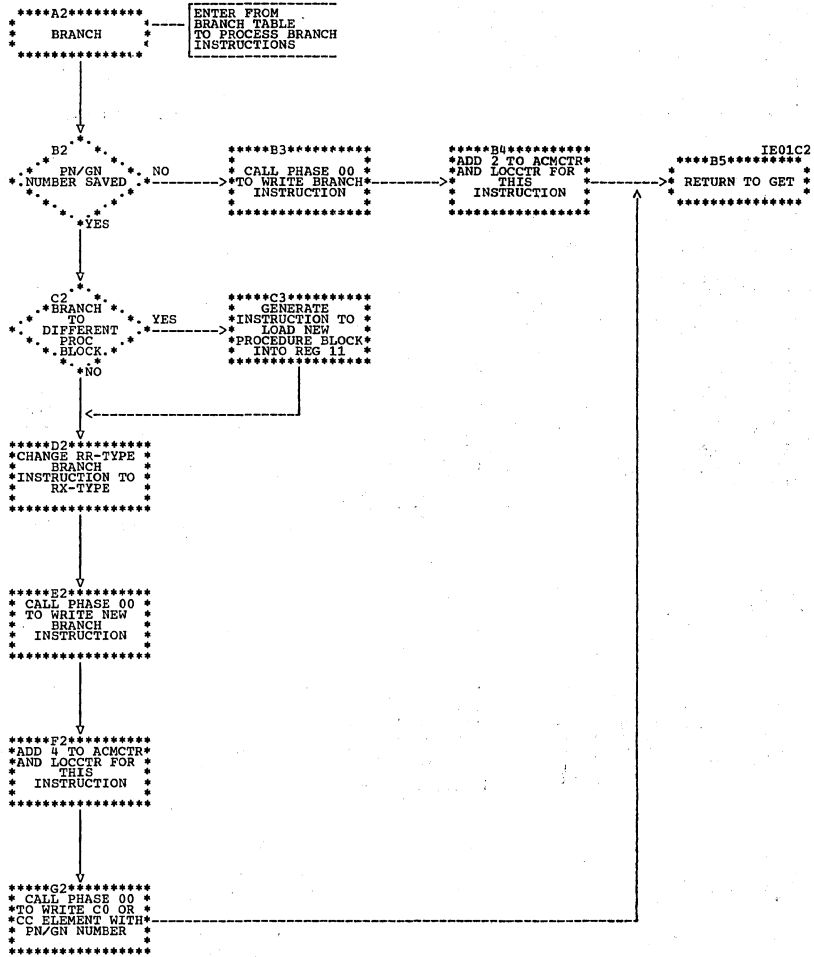




Chart IG. Phase 63: GNDEF Routine

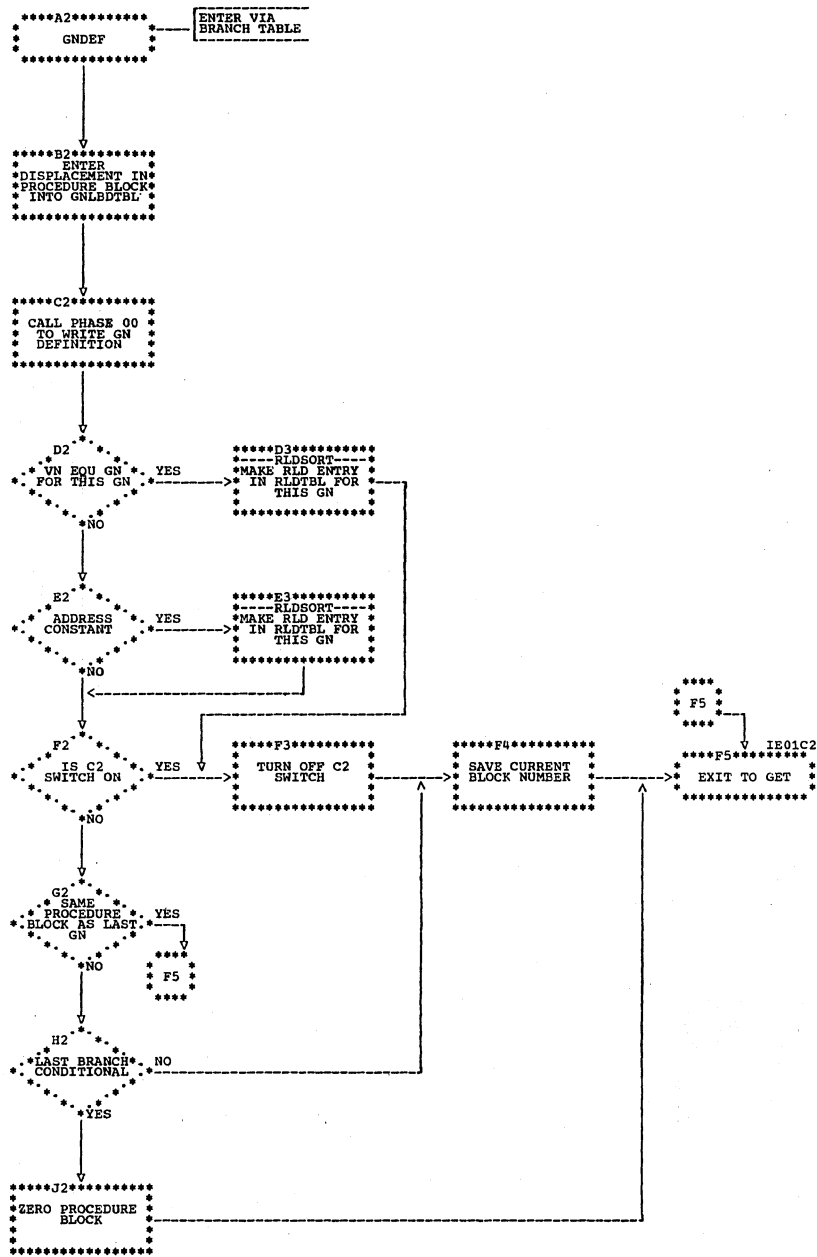


Chart IH. Phase 63: PNDEF Routine

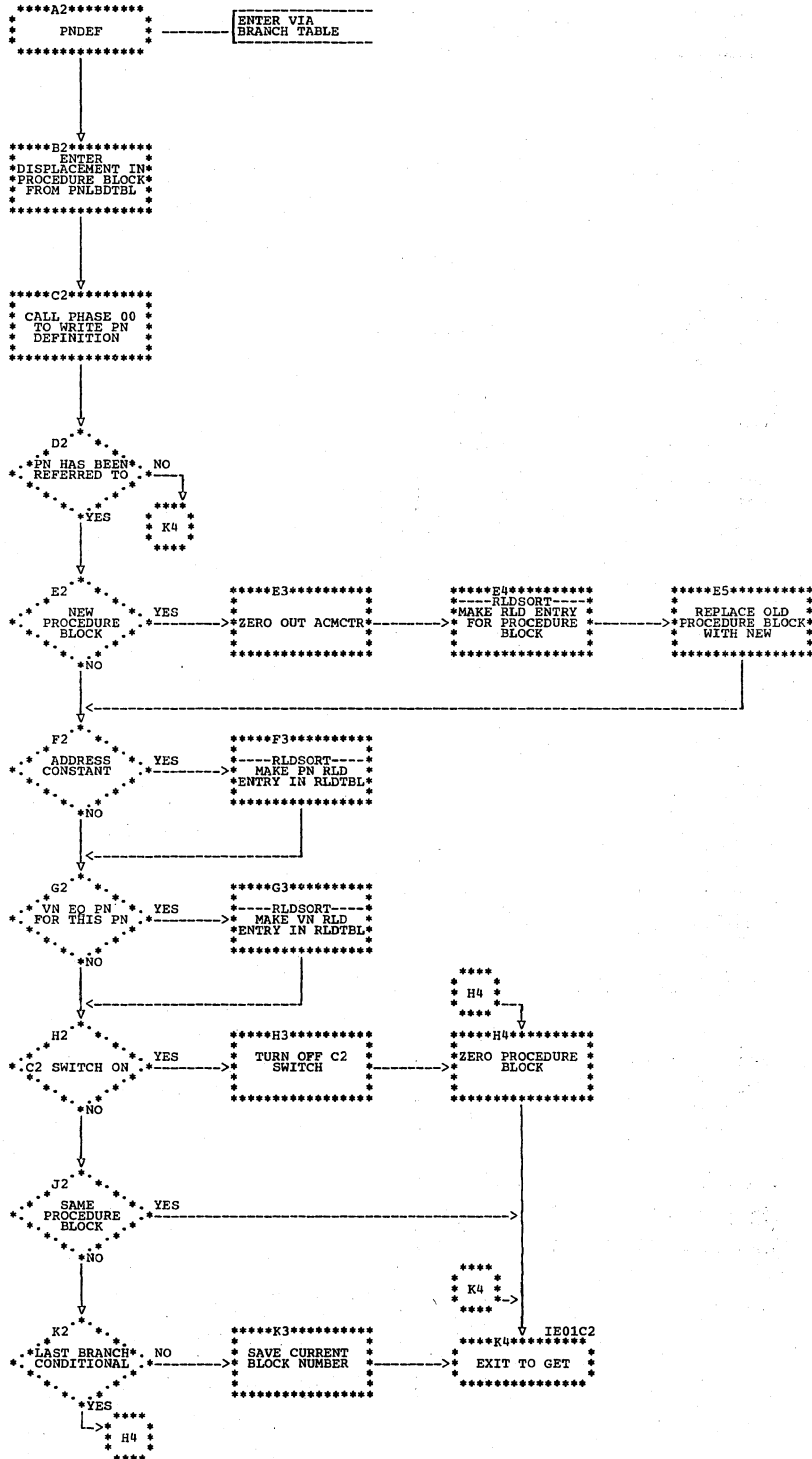


Chart II. Phase 63: ADREF Routine

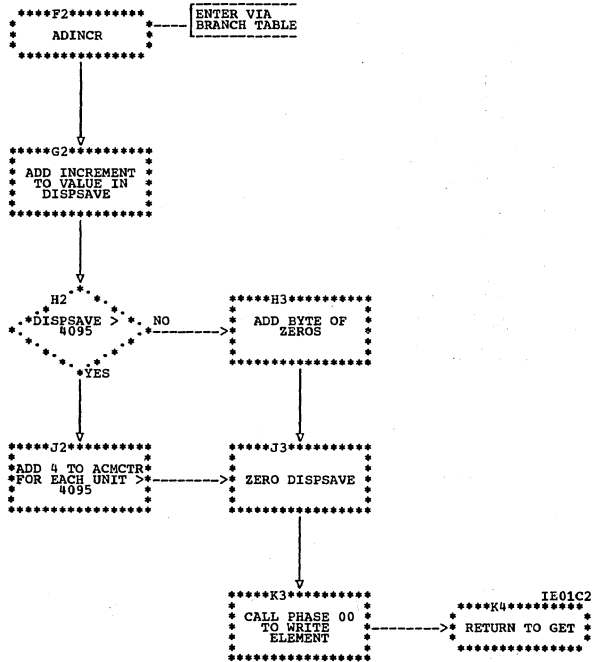
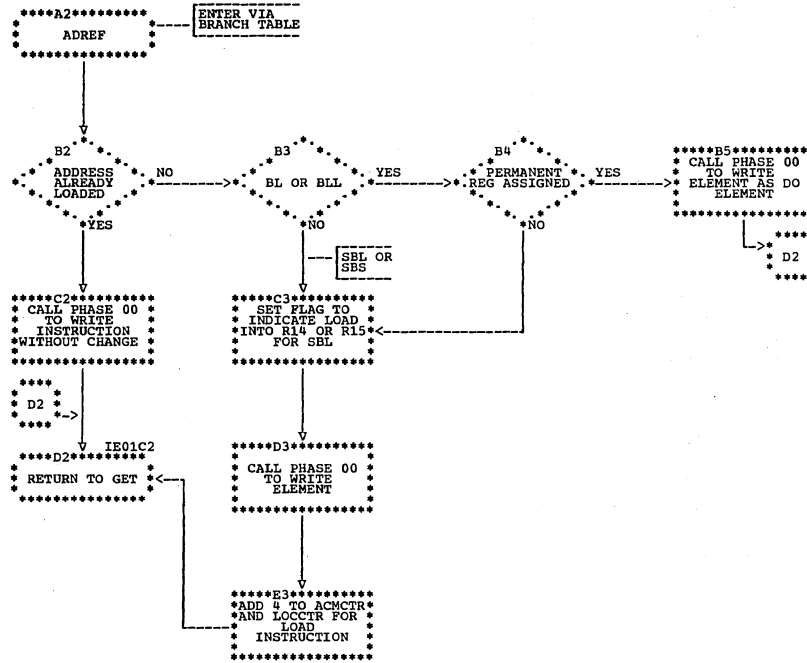


Chart IJ. Phase 63: C1REF Routine

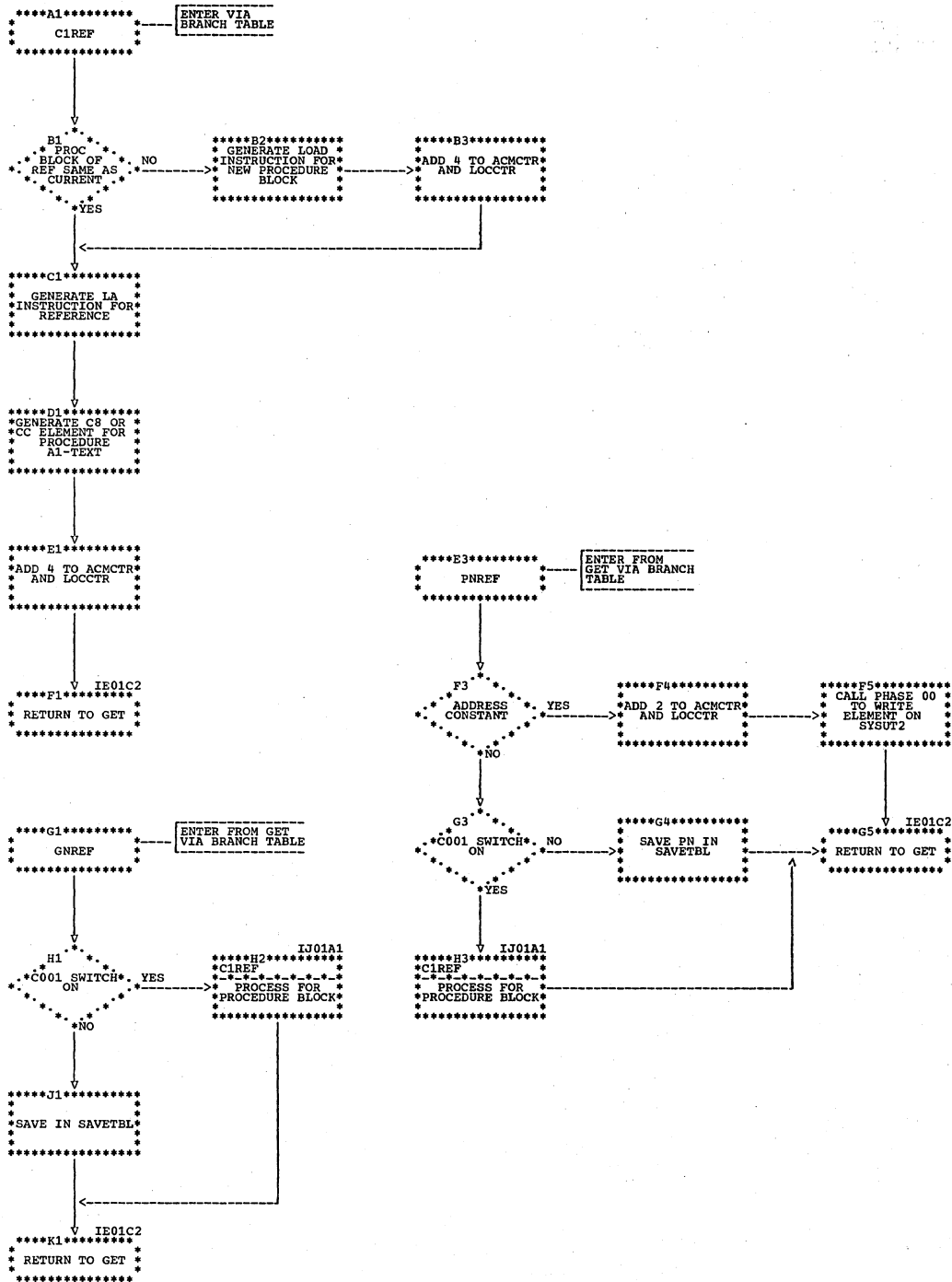


Chart IK. Phase 64: Overall Flow

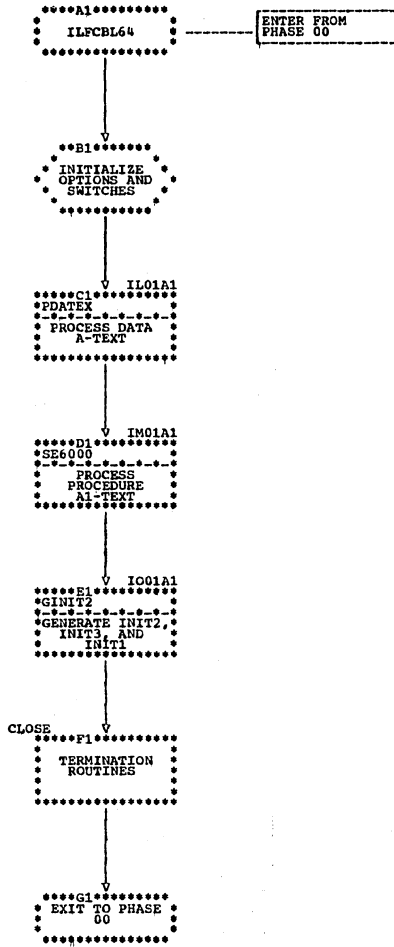


Chart II. Phase 64: PDATEX Routine

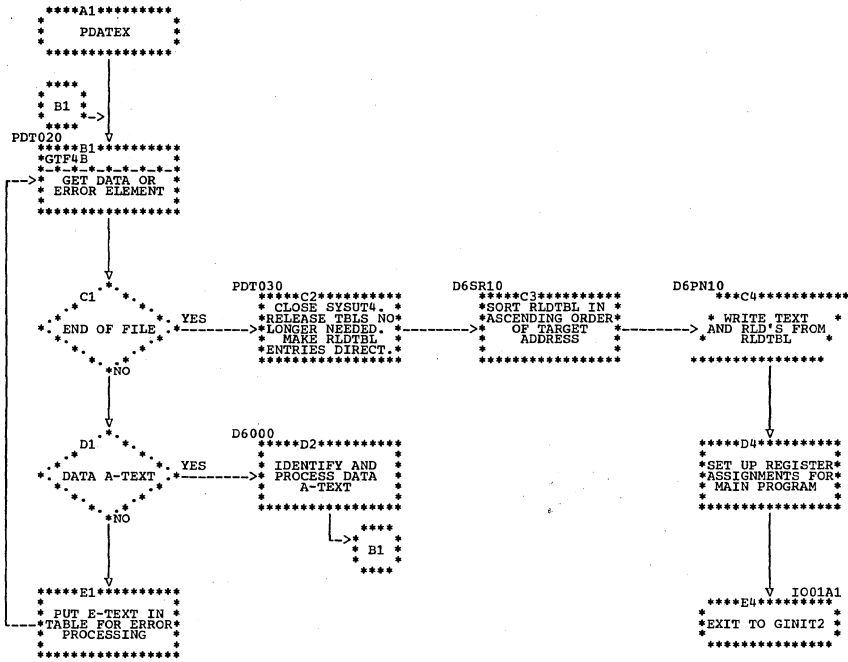


Chart IM. Phase 64: SE6000 Routine

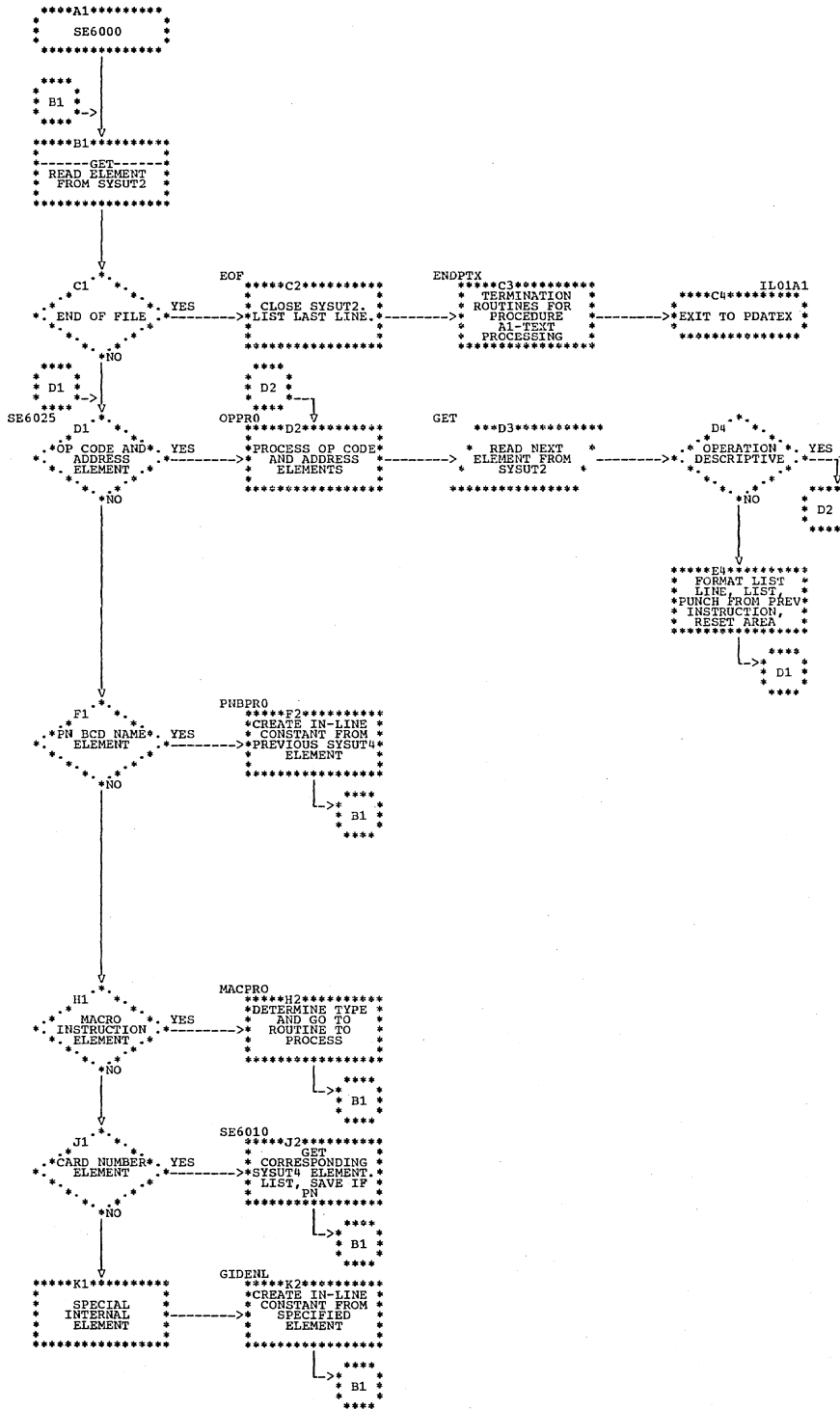


Chart IN. Phase 64: ADREF, RC4, RC8C, and RD001 Routines

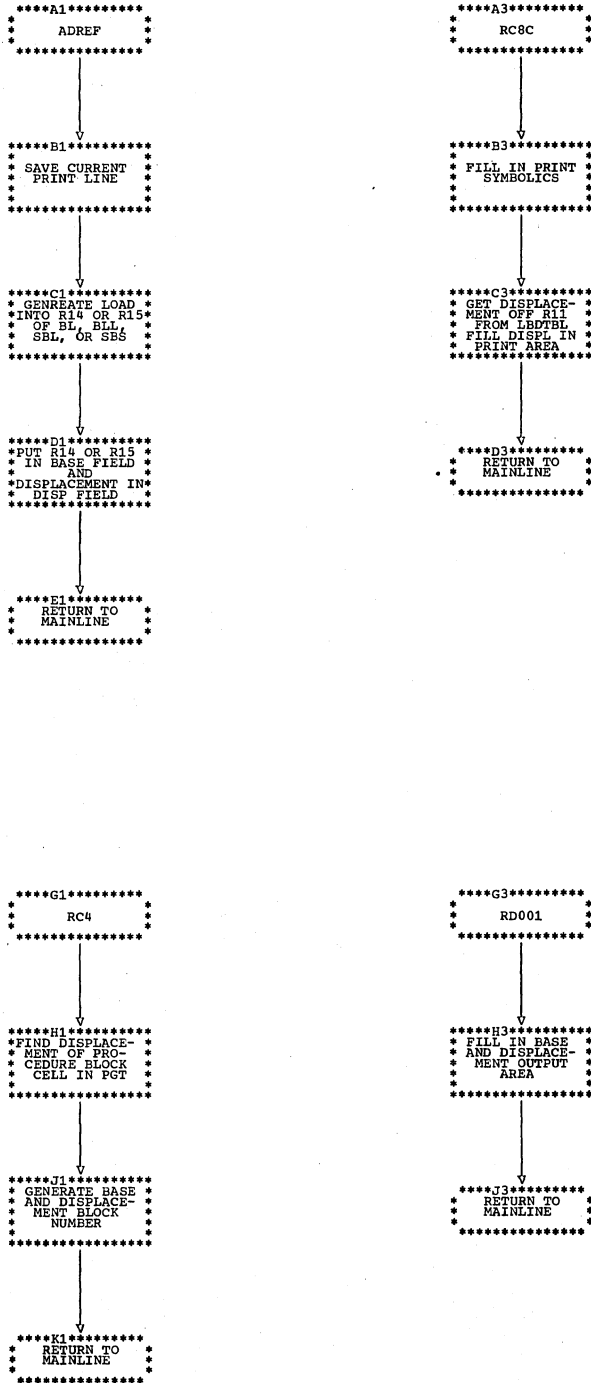




Chart IO. Phase 64: GINIT2 Routine

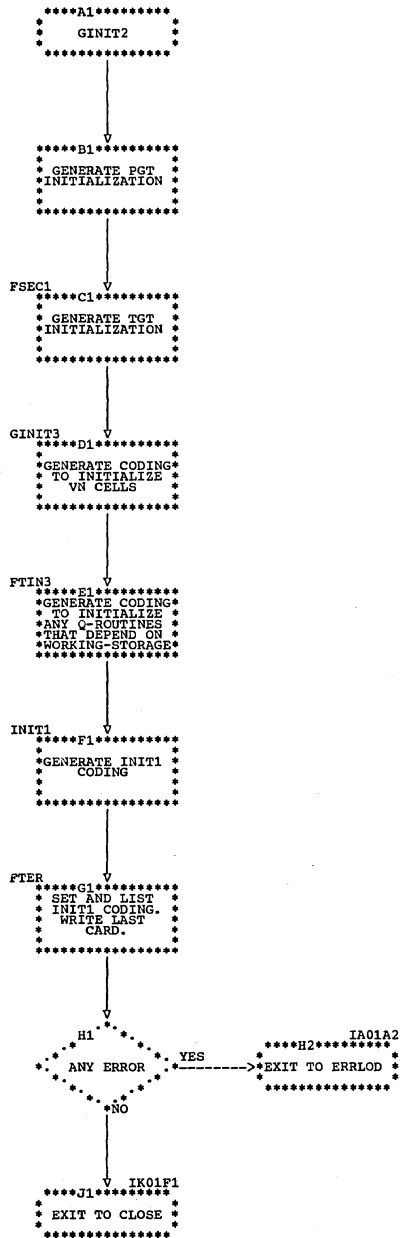


Chart IP. Phase 65: Overall Flow

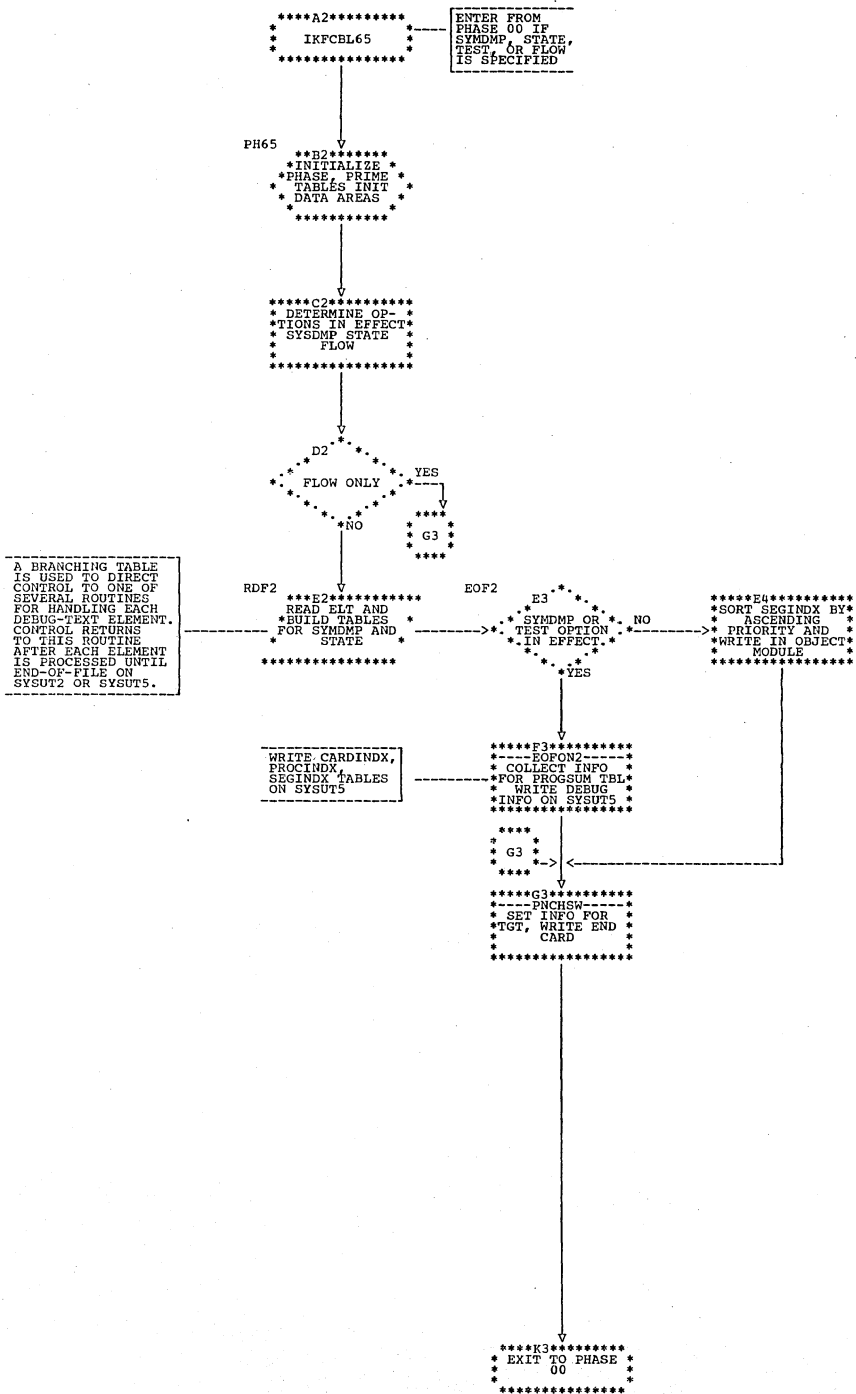


Chart IQ. Phase 65: TENPROC, TWENPROC, and GTEQ10K Routines

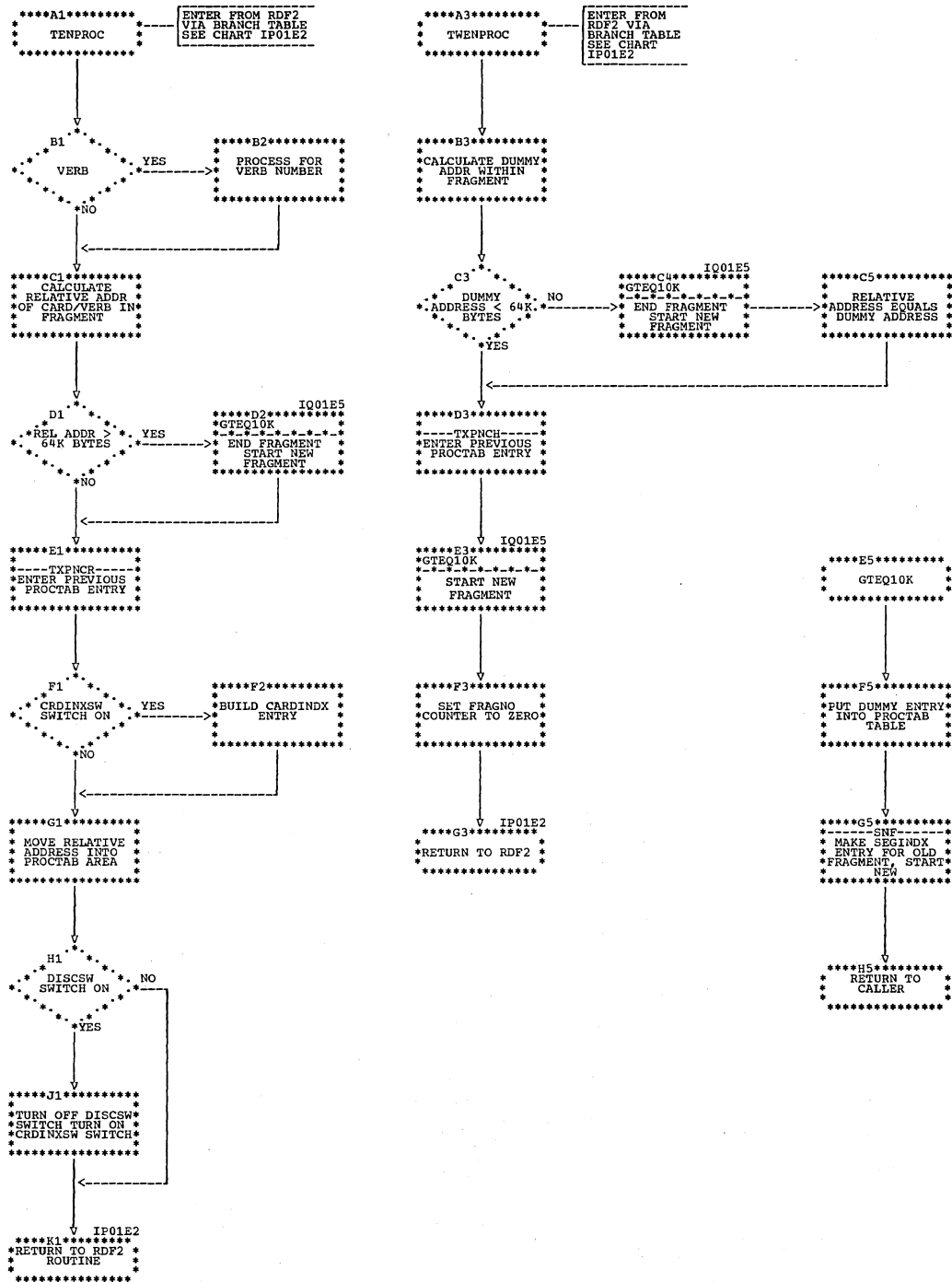


Chart IR (Part 1 of 3). Phase 6A: Overall Flow

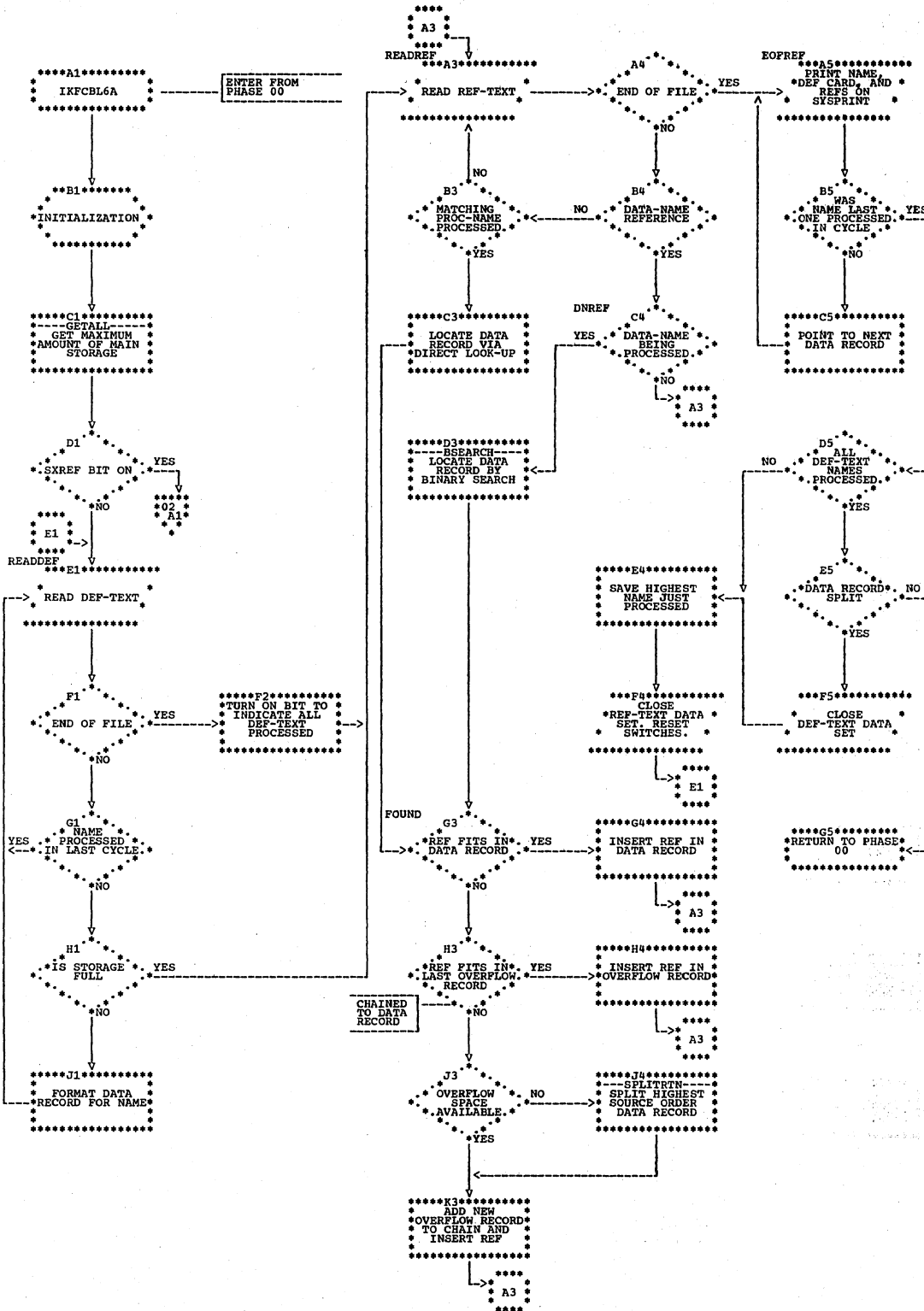


Chart IR (Part 2 of 3). Phase 6A: Overall Flow

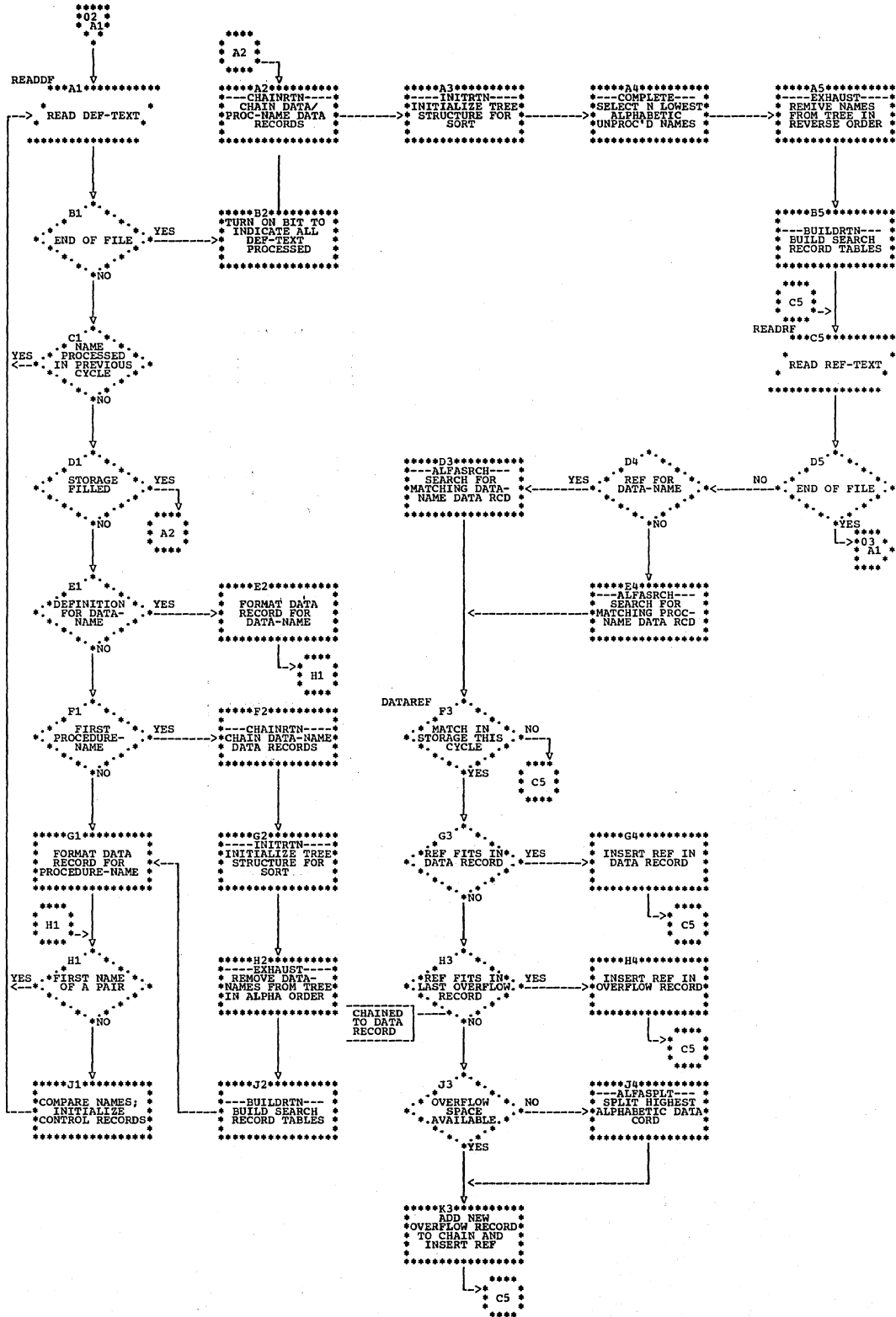


Chart IR (Part 3 of 3). Phase 6A: Overall Flow

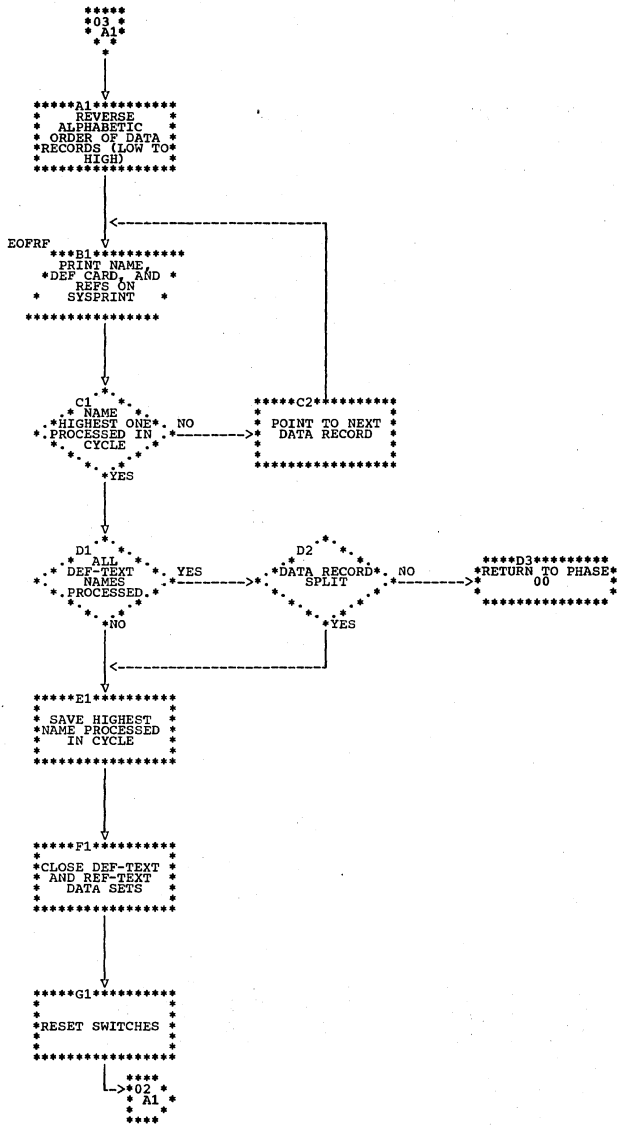
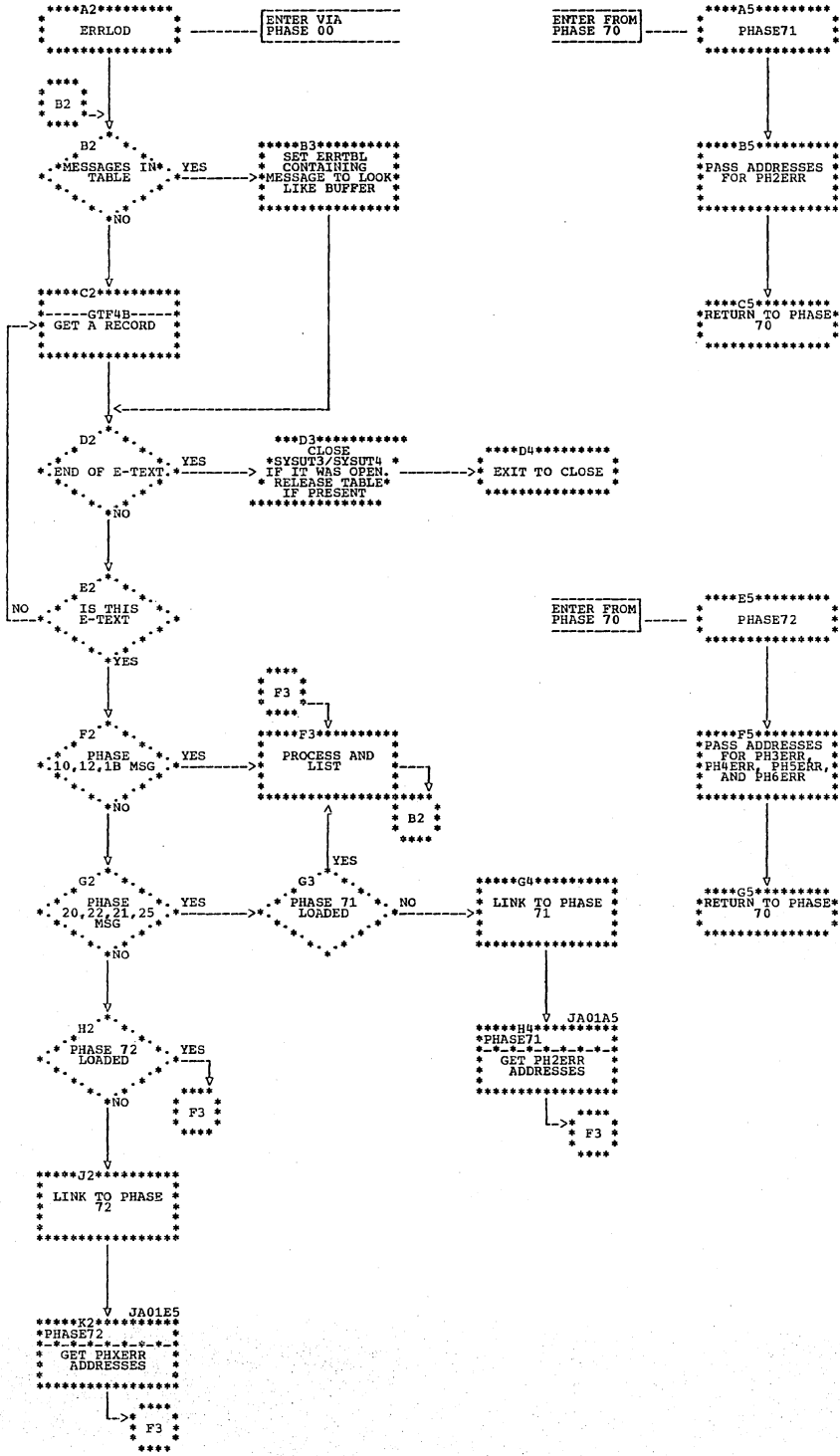
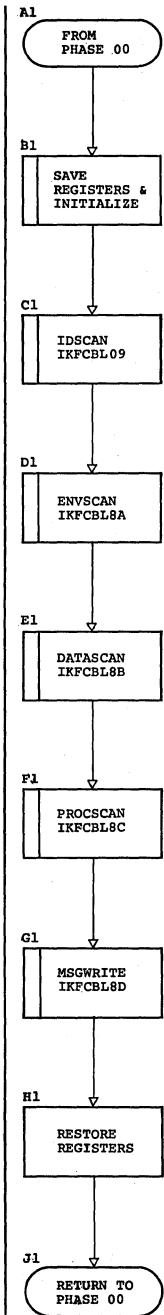


Chart JA. Phases 70, 71, and 72: Overall Flow



| Chart KA (Part 1 of 5). Phase 80: FIPS





( Chart KA (Part 2 of 5). Phase 80: FIPS

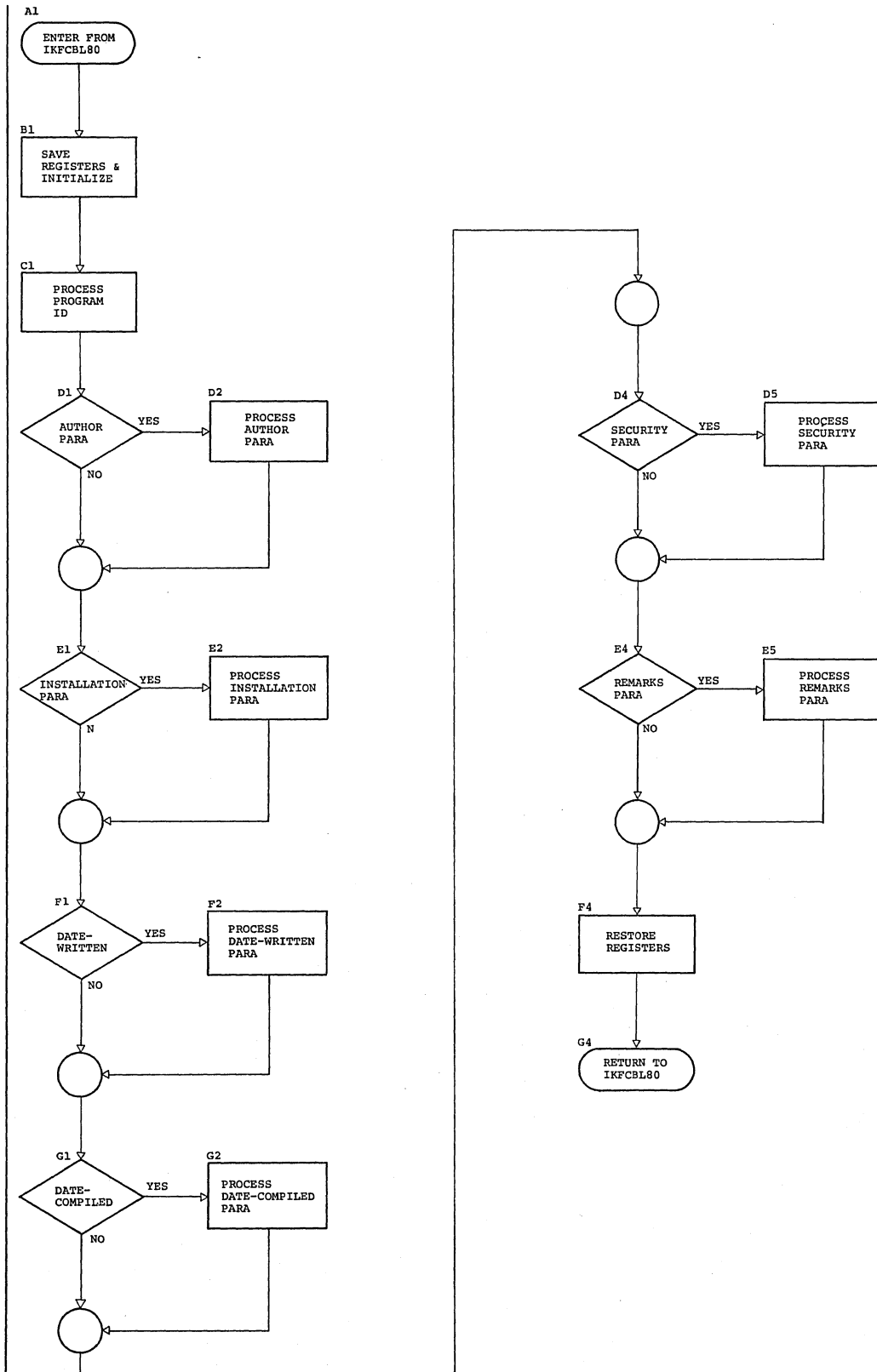


Chart KA (Part 3 of 5). Phase 80: FIPS

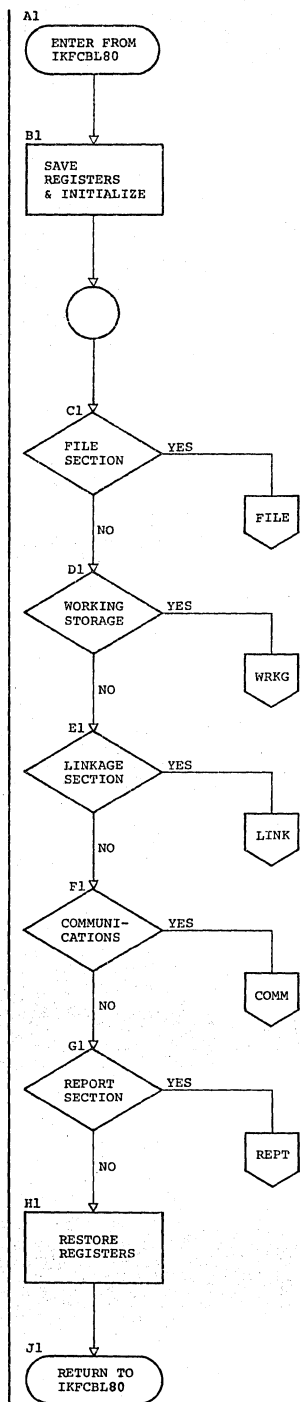


Chart KA (Part 4 of 5). Phase 80: FIPS

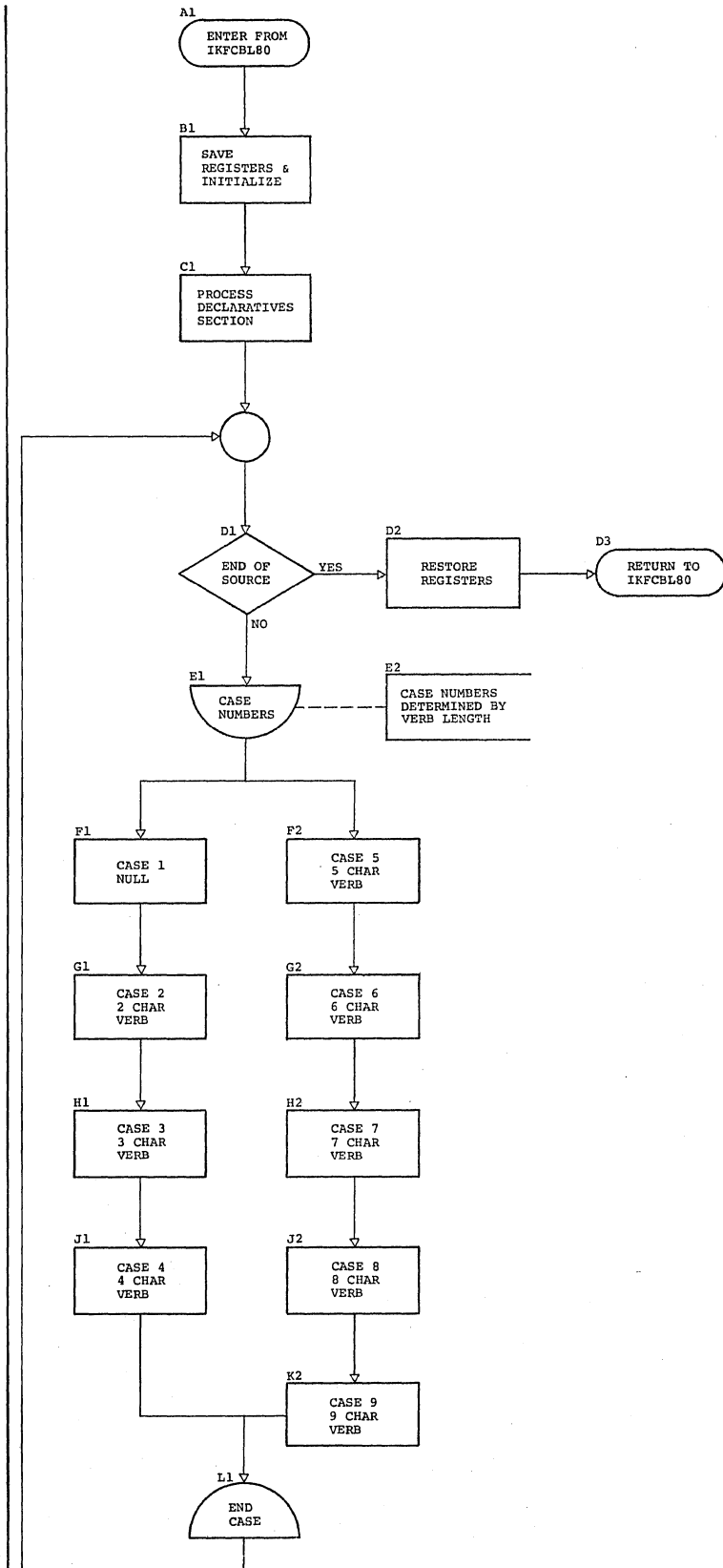
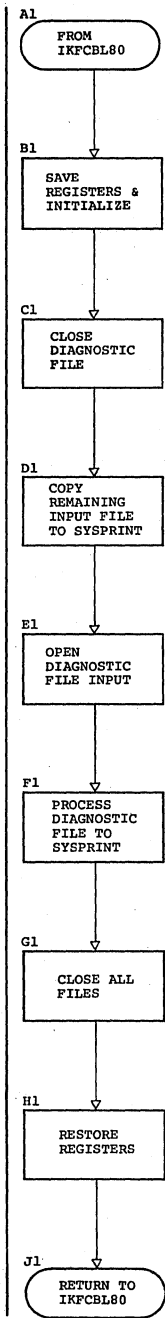


Chart KA (Part 5 of 5). Phase 80: FIPS



SECTION 4. DIRECTORY

FLOWCHART LABEL DIRECTORY

LABEL	CHART	DEFINITION		REFERENCE		LABEL	CHART	DEFINITION		REFERENCE	
		PAGE	BLOCK	PAGE	BLOCK			PAGE	BLOCK	PAGE	BLOCK
						DICTP1	DQ	01	H1	01	G1
						DIR	DE	01	D1	01	B1
										01	G1
ACCNET	DN	01	C3	01	C2	DIR010	DE	01	E1	01	D1
ANLZUFDS	ED					DLSCN	CK	01	D2	01	D1
ANLVBS	ED					DNREF	IR	01	C4	01	B4
						DOCODE	CF	01	E3	01	E2
BASISRTN	BD					DOCTL	CF	01	F3	01	F2
BEGIN	DA	01	B2	01	A2	DOLINE	CG	01	E3	01	E2
BEGIN	DM	01	B3	01	A3	DONGP	CI	01	B1	01	A1
BELEMI	DD	01	A4	01	J1	DOPAGE	CF	01	G3	01	G2
BGROUP	DD	01	J2	01	J1	DOROL	CJ	01	F1	01	D1
BLDOBO2	DP	01	C3	01	B3	DOUSGE	CG	01	G3	01	G2
BLDOBO6	DP	01	G3	01	F3	DUMTST	CA	01	B3	01	A3
BLDOBO7	DP	01	H3	01	G3	D6PN10	HE	01	C4	01	C3
BMBSRN	DD	01	C1	01	B1	D6PN10	IL	01	C4	01	C3
				01	B2	D6SR10	HE	01	C3	01	C2
BNORML	DD	01	D4	01	C4	D6SR10	IL	01	C3	01	C2
				01	C5	D6000	HE	01	D2	01	D1
				01	J3	D6000	IL	01	D2	01	D1
				01	J4						
BPASS12	DQ	01	F1	01	E1						
				01	E2	ELIPR	DL	01	J2	01	E3
BPASS2	DQ	01	G1	01	F1					01	H2
BREAD	DD	01	E1	01	D1	END	IC	01	E1	01	A5
				01	D2					01	B5
BSUBRN	DB	01	D3	01	C3					01	C5
				01	C4					01	D1
				01	D2					01	D5
BUSAGE	DD	01	G1	01	F1					01	E4
										01	E5
CDSCNA	CD	01	E3	01	E2					01	F2
CHKDCL	CK	01	D1	01	C1					01	G2
				01	C2					01	H2
CHKENT	EB	01	G3	01	F3	ENDPTX	HD	01	C3	01	C2
				01	F4	ENDPTX	ID	01	B2		
CHKPRNT	BC	01	F2	01	F1	ENDPTX	IM	01	C3	01	C2
				01	G1	ENDP13	DP	01	C5	01	A4
CLOSE	IA	01	E3	01	D3					01	B5
CLOSE	IK	01	F1	01	E1	END01A	CG	01	C3	01	C2
CLRD	CJ	01	H1	01	G1	END02	CH	01	E4	01	D4
COPYPROC	BD			01		END02A	CH	01	D4	01	D3
COPYRTN	BD					ENOPAT	HC	01	D3	01	D2
CTCDECK	GB	01	E1	01	D1	EOF	EA	01	E3	01	D3
CO	IE	01	F3	01	F2	EOF	FA	01	D4	01	D3
						EOF	HD	01	C2	01	C1
DATAREF	IR	02	F3	02	D3	EOF	IE	01	C3		
				02	E4	EOF	IM	01	C2	01	C1
DELIM	CK	01	E1	01	D1	EOFIN	GB	01	C2	01	C1
				01	D2	EOFREF	IR	01	A5	01	A4
				01	E3					01	C5
				01	G2	EOFRF	IR	03	B1	03	A1
				01	H3					03	C2
DICSCN	EB	01	B2	01	B1	EOF2	IP	01	E3	01	E2
				01	C2	EORDA	CF	01	D3	01	D2
				01	G3	EPFT	CI	01	C2	01	C1
DICTENTR	DR	01	C1	01	B1						
				01	K2						

Licensed Material - Property of IBM

LABEL	CHART	DEFINITION		REFERENCE		LABEL GETNXT	CHART DK	DEFINITION		REFERENCE	
		PAGE	BLOCK	PAGE	BLOCK			PAGE	BLOCK	PAGE	BLOCK
EPHD	CI	01	D2	01	D1						
ESD	HC	01	E4	01	E3					01	E2
FDSCN	CD	01	B3	01	B2					01	F4
FDTEXT	DB	01	C2	01	C1	GETOUT	DD	01	E4	01	G4
FILED	DQ	01	E3	01	E4	GETPTR	DJ	01	G4	01	D4
PLEX1	CI	01	C1	01	B1					01	G3
				01	B2	GIDENL	HD	01	K2	01	H3
FLVSCN	CD	01	B4	01	B3	GIDENL	IM	01	K2	01	K1
FORMLA	FB	01	D1	01	C1	GINIT3	HF	01	D1	01	K1
FOUND	IR	01	G3	01	C3	GINIT3	IO	01	D1	01	C1
				01	D3	GNEQUE	HC	01	G2	01	C1
FOURTY8	IE	01	E3	01	E2	GNGPLD	CJ	01	D1	01	G1
FSEND	DL	01	C3	01	C2	GNDLPT	CJ	01	B1	01	C1
FSEC1	HF	01	C1	01	B1	GNOPT	GD	01	K1	01	A1
FSEC1	IO	01	C1	01	B1					01	J1
FSTXT	DF	01	C3	01	C2	GNUMR	CJ	01	D3	01	J2
FSTXT	DN	01	C2	01	C1					01	C3
FST000	DL	01	F3	01	F2	GOTAVRNB	ED			01	C5
FST000	DL	01	C4	01	C3	GRIPR	DL	01	H3	01	H2
FTER	HF	01	G1	01	F1	GSPICR	DD	01	B2	01	B1
FTER	IO	01	G1	01	F1						
FTIN3	HF	01	E1	01	D1	HEADER	HB	01	C2	01	C1
FTIN3	IO	01	E1	01	D1	IDLHN	FA	01	F3	01	E3
F2EOF	HC	01	D2	01	D1	IFERR	FB	01	E1	01	E1
				01	H1	IF50	FB	01	F1	01	D1
GCKOP3	GD	01	D1	01	C1					01	E5
				01	J4					01	F5
GDOAGN	GD	01	J1	01	H1	INIT	DE	01	B1	01	A1
GDOAGN	GJ	01	H1	01	G1	INITL	BD				
GET	HD	01	D3	01	D2	INIT1	HF	01	F1	01	A1
GET	ID	01	B1	01	A1	INIT1	IO	01	F1	01	E1
				01	C4	INSROU	CJ	01	J1	01	H1
				01	D4	INSRT	HC	01	F3	01	H1
				01	E2	INSRTCHK	BD			01	F2
				01	F2	ISEND	FC	01	D2	01	C2
				01	H4	ITEMRN	DD	01	H1	01	G1
				01	K3						
				02	A2						
				02	B2	KOP3	GJ	01	D1	01	C1
				02	B3					01	J4
				02	D2						
				02	E2	LDSCN	CD	01	D3	01	D2
				02	E3	LHNAM	CK	01	E2	01	E1
				02	F1	LTLRTN	HC	01	F2	01	F1
				02	F2						
GET	IE	01	C2	01	B2	MACPRO	HD	01	H2	01	H1
				01	H2	MACPRO	IM	01	H2	01	H1
GET	IM	01	D3	01	D2	MACRO	IE	01	D3	01	D2
GETCRD	CE	01	C2	01	B2	MAYBE	BE				
GETDLM	CB	01	D2	01	C2						
GETDLM	CB	01	H4	01	H3	NOENDD	DQ	01	B1	01	A1
GETDLM	CD	01	C4	01	B4					01	A2
				01	C3					01	A3
				01	D3					01	D5
				01	F3					01	F2
				01	G2					01	G1
GETN	DP	01	D1	01	C1					01	G5
										01	H4

<u>LABEL</u>	<u>CHART</u>	<u>DEFINITION</u>		<u>REFERENCE</u>		<u>LABEL</u>	<u>CHART</u>	<u>DEFINITION</u>		<u>REFERENCE</u>	
		<u>PAGE</u>	<u>BLOCK</u>	<u>PAGE</u>	<u>BLOCK</u>			<u>PAGE</u>	<u>BLOCK</u>	<u>PAGE</u>	<u>BLOCK</u>
						<u>PNEQR</u>	<u>HC</u>	<u>C1</u>		<u>H01</u>	<u>H1</u>
							<u>CG</u>	<u>01</u>		<u>D01</u>	<u>D2</u>
<u>NOGET</u>	<u>DQ</u>	<u>01</u>	<u>E1</u>	<u>01</u>	<u>D1</u>	<u>PRTP</u>	<u>CG</u>	<u>01</u>	<u>F3</u>	<u>01</u>	<u>F2</u>
<u>NOTSTP</u>	<u>HB</u>	<u>01</u>	<u>F1</u>	<u>01</u>	<u>E1</u>	<u>PRINT</u>	<u>EB</u>	<u>01</u>	<u>D3</u>	<u>01</u>	<u>C3</u>
						<u>PRINT</u>	<u>EB</u>	<u>01</u>	<u>E3</u>	<u>01</u>	<u>D3</u>
<u>OD2FND</u>	<u>DP</u>	<u>01</u>	<u>F1</u>	<u>01</u>	<u>E1</u>					<u>01</u>	<u>C4</u>
<u>OKCOMP</u>	<u>FB</u>	<u>01</u>	<u>D4</u>	<u>01</u>	<u>D3</u>	<u>PROC77</u>	<u>DL</u>	<u>01</u>	<u>E3</u>	<u>01</u>	<u>E2</u>
				<u>01</u>	<u>E3</u>	<u>PR0110</u>	<u>CG</u>	<u>01</u>	<u>J2</u>	<u>01</u>	<u>H2</u>
<u>OPPRO</u>	<u>HD</u>	<u>01</u>	<u>D2</u>	<u>01</u>	<u>D1</u>	<u>PR02A</u>	<u>CG</u>	<u>01</u>	<u>H2</u>	<u>01</u>	<u>G2</u>
				<u>01</u>	<u>D4</u>	<u>PR02A</u>	<u>CH</u>	<u>01</u>	<u>D3</u>	<u>01</u>	<u>C3</u>
<u>OPPRO</u>	<u>ID</u>	<u>01</u>	<u>C2</u>	<u>01</u>	<u>C1</u>	<u>PR0250</u>	<u>CH</u>	<u>01</u>	<u>G3</u>	<u>01</u>	<u>F3</u>
				<u>01</u>	<u>K2</u>	<u>PUNCH</u>	<u>IC</u>	<u>01</u>	<u>F3</u>	<u>01</u>	<u>D1</u>
<u>OPPRO</u>	<u>IM</u>	<u>01</u>	<u>D2</u>	<u>01</u>	<u>D1</u>	<u>PUTEQU</u>	<u>GI</u>	<u>01</u>	<u>F4</u>	<u>01</u>	<u>D4</u>
				<u>01</u>	<u>D4</u>					<u>01</u>	<u>E3</u>
										<u>01</u>	<u>G1</u>
<u>PDSCN</u>	<u>CK</u>	<u>01</u>	<u>B1</u>	<u>01</u>	<u>A1</u>						
<u>PDT020</u>	<u>HE</u>	<u>01</u>	<u>B1</u>	<u>01</u>	<u>A1</u>						
				<u>01</u>	<u>D2</u>	<u>QIFOUND</u>	<u>DP</u>	<u>01</u>	<u>G1</u>	<u>01</u>	<u>F1</u>
				<u>01</u>	<u>E1</u>	<u>QUAL</u>	<u>DJ</u>	<u>01</u>	<u>H3</u>	<u>01</u>	<u>G3</u>
<u>PD T020</u>	<u>IL</u>	<u>01</u>	<u>B1</u>	<u>01</u>	<u>A1</u>	<u>QUAL</u>	<u>DI</u>	<u>01</u>	<u>C4</u>	<u>01</u>	<u>C3</u>
				<u>01</u>	<u>E1</u>	<u>RDF2</u>	<u>IP</u>	<u>01</u>	<u>E2</u>	<u>01</u>	<u>D2</u>
				<u>01</u>	<u>D2</u>	<u>RDPERD</u>	<u>CF</u>	<u>01</u>	<u>C2</u>	<u>01</u>	<u>B2</u>
<u>PD T030</u>	<u>HE</u>	<u>01</u>	<u>C2</u>	<u>01</u>	<u>C1</u>					<u>01</u>	<u>E3</u>
<u>PDT030</u>	<u>IL</u>	<u>01</u>	<u>C2</u>	<u>01</u>	<u>C1</u>					<u>01</u>	<u>F3</u>
<u>PERD01</u>	<u>CG</u>	<u>01</u>	<u>B2</u>	<u>01</u>	<u>A2</u>					<u>01</u>	<u>G3</u>
				<u>01</u>	<u>D3</u>	<u>RDSYN</u>	<u>DL</u>	<u>01</u>	<u>D4</u>	<u>01</u>	<u>D3</u>
				<u>01</u>	<u>E3</u>	<u>READ</u>	<u>AA</u>	<u>02</u>	<u>B1</u>	<u>01</u>	<u>C1</u>
				<u>01</u>	<u>F4</u>	<u>READB</u>	<u>AA</u>	<u>02</u>	<u>C1</u>	<u>02</u>	<u>B1</u>
				<u>01</u>	<u>G3</u>					<u>06</u>	<u>B2</u>
				<u>01</u>	<u>J2</u>	<u>READC</u>	<u>AA</u>	<u>02</u>	<u>D1</u>	<u>02</u>	<u>C1</u>
<u>PERD02</u>	<u>CH</u>	<u>01</u>	<u>B3</u>	<u>01</u>	<u>A2</u>	<u>READDEF</u>	<u>IR</u>	<u>01</u>	<u>E1</u>	<u>01</u>	<u>D1</u>
				<u>01</u>	<u>B2</u>					<u>01</u>	<u>F4</u>
<u>PG TINT</u>	<u>HB</u>	<u>01</u>	<u>E1</u>	<u>01</u>	<u>D1</u>					<u>01</u>	<u>G1</u>
<u>PGTINT</u>	<u>IB</u>	<u>01</u>	<u>G3</u>	<u>01</u>	<u>F3</u>					<u>01</u>	<u>J1</u>
<u>PHCTRL</u>						<u>READDF</u>	<u>IR</u>	<u>02</u>	<u>A1</u>	<u>01</u>	<u>D1</u>
<u>PHINIT</u>	<u>CE</u>	<u>01</u>	<u>B2</u>	<u>01</u>	<u>A2</u>					<u>02</u>	<u>C1</u>
<u>PHINIT</u>	<u>FA</u>	<u>01</u>	<u>B3</u>	<u>01</u>	<u>A3</u>					<u>02</u>	<u>H1</u>
<u>PHTERM</u>	<u>DA</u>	<u>01</u>	<u>F2</u>	<u>01</u>	<u>G2</u>					<u>02</u>	<u>J1</u>
				<u>01</u>	<u>G3</u>					<u>03</u>	<u>G1</u>
<u>PHTERM</u>	<u>DM</u>	<u>01</u>	<u>D3</u>	<u>01</u>	<u>C3</u>	<u>READF2</u>	<u>HC</u>	<u>01</u>	<u>C1</u>	<u>01</u>	<u>B1</u>
<u>PH03</u>	<u>BC</u>	<u>01</u>	<u>B1</u>	<u>01</u>	<u>A1</u>					<u>01</u>	<u>E4</u>
<u>PH5BVB</u>	<u>CA</u>	<u>01</u>	<u>G2</u>	<u>01</u>	<u>F2</u>					<u>01</u>	<u>F3</u>
<u>PH5CTL</u>	<u>GA</u>	<u>01</u>	<u>B2</u>	<u>01</u>	<u>A2</u>					<u>01</u>	<u>G2</u>
				<u>01</u>	<u>D3</u>					<u>01</u>	<u>H2</u>
				<u>01</u>	<u>F4</u>	<u>READF2</u>	<u>IC</u>	<u>01</u>	<u>C1</u>	<u>01</u>	<u>B1</u>
				<u>01</u>	<u>G4</u>					<u>01</u>	<u>E1</u>
				<u>01</u>	<u>J3</u>	<u>READLIB</u>	<u>AA</u>	<u>06</u>	<u>A2</u>	<u>02</u>	<u>D2</u>
				<u>01</u>	<u>J4</u>	<u>READREF</u>	<u>IR</u>	<u>01</u>	<u>A3</u>	<u>01</u>	<u>B3</u>
<u>PH65</u>	<u>IP</u>	<u>01</u>	<u>B2</u>	<u>01</u>	<u>A2</u>					<u>01</u>	<u>C4</u>
<u>PICTAN</u>	<u>DD</u>	<u>01</u>	<u>B5</u>	<u>01</u>	<u>B4</u>					<u>01</u>	<u>F2</u>
<u>PLSCALL</u>	<u>AA</u>									<u>01</u>	<u>G4</u>
<u>PLUS1</u>	<u>GD</u>	<u>01</u>	<u>H1</u>	<u>01</u>	<u>G1</u>					<u>01</u>	<u>H1</u>
<u>PLUS1</u>	<u>GJ</u>	<u>01</u>	<u>G1</u>	<u>01</u>	<u>F1</u>					<u>01</u>	<u>H4</u>
<u>PNBPRO</u>	<u>HD</u>	<u>01</u>	<u>F2</u>	<u>01</u>	<u>F1</u>					<u>01</u>	<u>K3</u>
<u>PNBPRO</u>	<u>IM</u>	<u>01</u>	<u>F2</u>	<u>01</u>	<u>F1</u>						
<u>PNDEFRTN</u>	<u>ED</u>	<u>01</u>	<u>E5</u>	<u>03</u>	<u>C2</u>						
				<u>04</u>	<u>A1</u>						

LABEL	CHART	DEFINITION		REFERENCE		LABEL	CHART	DEFINITION		REFERENCE	
		PAGE	BLOCK	PAGE	BLOCK			PAGE	BLOCK	PAGE	BLOCK
READRF	IR	02	C5	02	B5	THRESUBS	DR	01	C4		
				02	F3	TRMNATE	AA	06	B4	06	A4
				02	G4					06	G2
				02	H4	TRMNATE	BC	01	F1	01	D5
				02	K3					01	E1
REDEF	DL	01	D3	01	D2	TWOSUBS	DR	01	C3		
RELEASE	IC	01	H3	01	G3	TXPNH	HC	01	E3	01	E2
RENAMS	DK	01	F4	01	F3	TXPNH	IC	01	G3	01	F3
RENM10	DQ	01	J4	01	J3						
REPORTD	DQ	01	G5	01	G4						
RETURN	AA	01	G2	01	G1	VALGEN	DD	01	C5	01	C4
RLIBA	AA	06	E2	06	C2	VALGEN	DD	01	J4	01	J3
				06	D2	VIRRTN	HC	01	E2	01	E1
				06	D3	VNDEPR	HD	01	E2	01	E1
RLIBD	AA	06	A4	06	A2	VRBSCN	CK	01	F3	01	F1
RLIBGO	AA	06	H2	06	G2	WGO	AA	05	C2	01	C3
SDTEXT	DB	01	D2	01	D1					05	B2
SDTEXT	DN	01	D2	01	D1					05	C3
SDTXT	DF	01	D3	01	D2					05	D4
SE6010	HD	01	J2	01	J1	WLVSCN	CD	01	C3	01	C2
SE6010	IM	01	J2	01	J1	WOUT	AA	04	D1	01	A3
SE6025	HD	01	D1	01	C1					04	C1
				01	E4	WPCH	AA	05	B2	01	B3
SE6025	IM	01	D1	01	C1	WRITE	AA	04	C1	01	B1
				01	E4					04	E1
SKPRNT	BC	01	G3	01	F3	WRITEA	AA	04	B1	01	D1
				01	F4					03	D2
SKPRNTA	BC	01	F3	01	F2					03	E1
				01	G2	WRITES	DQ	01	B2		
SKPRNT3	BC	01	H3	01	G3	WRTEER	BC	01	F4	01	H2
SORTREN	DO	01	C3	01	C2					01	F3
SRCHTB	DD	01	D2	01	D1						
STEP1	BC	01	C1	01	B1						
STEP2	BC	01	D1	01	C1	XITXIT	CI	01	C4	01	A5
										01	B4
										01	B5
TERM	DE	01	E2	01	E1					01	C2
TESTSB2	DR	01	D2	01	C2					01	D2
				01	C3					01	E2
				01	C4					01	F2
TESTSB3	DR	01	F2	01	D3					01	G2
				01	E2					01	H2
TESTSB4	DR	01	H2	01	G2					01	J2
				01	G3	XIT1	CI	01	F1	01	E1
TESTSB6	DR	01	D3	01	D2	XIT2	CI	01	G1	01	F1
				01	K3	XIT2A	CI	01	G2	01	G1
TESTSB8	DR	01	J3	01	J2	XIT3	CI	01	H1	01	G1
TGTINT	HB	01	D1	01	C1	XIT4	CI	01	J1	01	H1
				01	C2	XIT5	CI	01	A4	01	J1
TGTINT	IB	01	F3	01	E3	XIT6	CI	01	B5	01	B4
				01	E4	XIT7	CI	01	A5	01	A4



TABLES USED BY PHASES

Phase	Table and TIB Number	
	Built or Changed by Phase	Referenced Only
10	ALPHTBL (27), CKPTBL (8), ENVTBL (3), FNTBL (10), HASH (30), INDTBL (4), INDXTB (34), KEYTAB (26), OD2TBL (9), PIOTBL (7), P1BTBL (2), QLTABL (1), QNMTBL (2), RCDTBL (11), RWRTBL (13), SATBL (5), SPNTBL (21), SRATBL (6), TOTTBL (32), UPSTBL (29)	
12	CTLTBL (14), DETTBL (17), GCNTBL (24), HASH (30), NPTTBL (18), PIOTBL (7), P1BTBL (2), QALTBL (23), QLTABL (1), RNMTBL (12), ROLTBL (15), ROUTBL (16), RWRTBL (13), SMSTBL (28), SNMTBL (35), SRCTBL (22), SUMTBL (19)	FNTBL (10), SPNTBL (21)
1B	DICOT (20), GVFNTBL (4), GVNMTBL (3), HASH (30), PIOTBL (7), PNQTBL (6), PNTABL (5), QLTABL (1), RNMTBL (12), USETBL (26), VRDEFTBL (14)	ALPHTBL (27), DETTBL (17), FNTBL (10), PIBTBL (2), P1BTBL (2), RCDTBL (11), ROUTBL (16), RWRTBL (13), SPNTBL (21)
20	VALGRP (6), VALTRU (33), LABTBL (13)	OD2TBL (9)
22	DICOT (20), FDTAB (28), GPLSTK (10), HASH (30), INDKEY (31), MASTODO (13), OBJSUB (5), OCCTBL (2), QFILE (23), QITBL (22), QRTN (21), QSBL (25), QVAR (24), RDFSTK (11), RENAMTB (3), RNMTBL (12), SRCHKY (34), VARLTBL (15)	OD2TBL (9), TOTTBL (32), UPSTBL (29), VALGRP (6), VALTRU (33)
21	CKPTBL (8), DATATAB (16), HASH (30), IND2TBL (17), QFILE (23), RUNTBL (35), SAMETB (19), SMRCDTBL (5)	FDTAB (28), DICOT (20), PIOTBL (7)
25	OCCTBL (2), ODOTBL (14)	DICOT (20), HASH (30), MASTODO (13), OD2TBL (9), QITBL (22), QRTN (21), RENAMTB (3), VARLTBL (15)
3	DTAB (4), QFILE (23), QVAR (24), USNGTBL (28)	DICOT (20), HASH (30), INDKEY (31), IND2TBL (17), VALTRU (33), QSBL (25)
35	DBGXT (6), DTAB (4), PITEXT (5), VRBDN (7)	

Figure 59. Tables Used by Phases (Part 1 of 2)

Table and TIB Number		
Phase	Built or Changed by Phase	Referenced Only
4	DBGTBL (13), DEFSEBS (18), KEYTBL (20), PFMTBL (12), PNOUNT (14), PSHTBL (17), PSIGNT (15), PTRFLS (16), SETTBL (21), STRING (9), VARYTB (1), VNTBL (11)	
45	SSCIN (5), SSCOUT (11), SSDELIM (20), TXTOUT (19)	
50	BLUSTBL (10), XAVAL (2), XINTR (1), XSCRPT (3), XSSNT (4)	
51	BLUSTBL (10), GNCAL TBL (16), PNU TBL (6), SEG TBL (15)	ALPHTBL (27), RUNTBL (35), USETBL (26)
6	CONDIS (14), CONTBL (9), CVIRTB (12), ERRTBL (10), GNTBL (8), LTLTBL (4), PNTBL (7), QTBL (3), RLDTBL (none), TGTADTBL (1), VIRPTR (13), VNPTY (17)	PNU TBL (6), SEG TBL (15)
62	BLASGTBL (16), BLVNTBL (23), CONDIS (14), CONTBL (9), CVIRTB (12), DRPLTBL (25), DRPTBL (24), GNATBL (8), GNFWD B TB (21), GNLABTBL (19), LTLTBL (4), PNATBL (7), PNFWD B TB (20), PNLABTBL (18), TGTADTBL (1), VIRPTR (13), VNPNTBL (29), VNPTY (17)	BLUSTBL (10), PNU TBL (6), SEG TBL (15)
63	GNLBDTBL (27), PNLBDTBL (26), QGNTBL (24), RLDTBL (28), TGTADTBL (1), VNPTY (17)	BLASGTBL (16), BLVNTBL (23), DRPLTBL (25), GNATBL (8), GNLABTBL (19), PNATBL (7), PNLABTBL (18), SEG TBL (15), VNPNTBL (29)
64	ERRTBL (10), QTBL (3), RLDTBL (28), TGTADTBL (1)	BLASGTBL (16), GNATBL (8), GNLBDTBL (27), LTLTBL (4), PNATBL (7), PNLBDTBL (26), QGNTBL (24), VIRPTR (13), VNPTY (17)
65	CARDINDX (11), PROCINDX (5), SEGINDX (16)	TGTADTBL (1)
6A	CNTLTBL (none), DATATBL (none), OFLCTBL (none)	
70		ERRTBL (10)

Figure 59. Tables Used by Phases (Part 2 of 2)

Figure 60. TIB Usage

TIB Number	Processing Phases																			
	10	12	18	20	22	21	25	3	35	4	45	50	51	6	62	63	64	65	6A	70
0																				
1	QLTABL									VARYTB		XINTR		TGTABTBL	TGTABTBL					
2	PIBTBL*				OCCTBL							XAVAl								
3	ENVTBL		GVMNTBL		RENAMTB							XSCRPT		QTBL			QTBL			
4	INDTBL		GVFNTBL					DTAB	DTAB			XSSNT		LTLTBL	LTLTBL					
5	SATBL		PNTABL		OBJSUB	SMRCDTBL			PITEXT		SSCIN							PROCINDX		
6	SRATBL		PNQDTBL	VALGRP					DBGTXT				PNUTBL							
7	PIOTBL								VRBDN					PNTBL	PNATBL					
8	CKPTBL													GNTBL	GNATBL					
9	OD2TBL									STRING				CONTBL	CONTBL					
10	FNTBL				GPLSTK							BLUSTBL					ERRTBL		ERRTBL	
11	RCDTBL				RDFSTK					VNTBL	SSCOUT								CARDINDX	
12		RNMTBL			RNMTBL					PFMTBL				CVIRTB	CVIRTB					
13	RWRTBL			LABTBL	MASTODO					DBGTBL				VIRPTR	VIRPTR					
14		CTLTBL					ODOTBL			PNOUNT				CONDIS	CONDIS					
15		ROLTBL			VARLTBL					PSIGNT			SEGTBL							
16		ROUTBL				DATATAB				PTRFLS			GNCALTBL		BLASGTBL			SEGINDX		
17		DETTBL								PSHTBL				VNPTY						
18		NPTTBL								DEFSBS					PNLABTBL					
19		SUMTBL				SAMETB					TXTOUT				GNLABTBL					
20			DICOT							KEYTBL	SSEDELIM				PNFWDBTB					
21	SPNTBL				QRN					SETTBL					GNFWDBTB					
22		SRCTBL			QITBL															
23		QALTBL			QFILE										BLVNTBL					
24		GCNTBL			QVAR										DRPTBL	QGNTBL				
25					QSBL										DRPLTBL					
26	KEYTAB																			
27	ALPHTBL		USETBL										ALPHTBL				PNLBDTBL			
28		SMSTBL			FDTAB			USNGTBL									PNLBDTBL			
29	UPSTBL				UPSTBL										VNPNPTBL					
30	HASH																			
31					INDKEY															
32	TOTTBL																			
33				VALTRU																
34	INDXTB				SRCHKY															
35		SNMTBL				RUNTBL														

Legend:  
 \*The QNMTBL table also uses TIB 2 during phase 10 processing.  
 \*\*The REPTAB table (TIB 29) is used only during Phase 02.  
 Note that the arrowhead indicates the last phase to process the table. Where more than one arrowhead follows a table name, the last phase is either phase 6 or one of the optimizer phases.

	IC (Internal Compiler)	A (Assembler)	E (Error)	XREF-Text	Debug-Text
Phase 04			E-text		
Phase 10	Data IC-text		E-text		
Phase 12	Data IC-text Procedure IC-text (P0-text for Report Writer subprogram)		E-text		
Phase 1B	Procedure IC-text (P0-text for Procedure Division)		E-text		
Phase 20	ATF-text	Data A-text (incomplete)	E-text		
Phase 22	Procedure IC-text (P0-text for Q-Routines)	Data A-text	E-text	DEF-text (for data-names)	
Phase 21		Data A-text	E-text		
Phase 3	Procedure IC-text (P1-form)		E-text	DEF-text (for procedure-names)	
Phase 35	Procedure IC-text (PIA-form)		E-text		
Phase 4	Procedure IC-text (P2-form) ATM-text		E-text		
Phase 45	Procedure IC-text (P2-text for UNSTRING verb)		E-text		
Phase 50	Procedure IC-text (P2-form)	Intermediate Procedure A-text Intermediate Optimization A-text	E-text Inter- mediate E-text		
Phase 51		Final Procedure A-text Final Optimization A-text	E-text		
Phase 6				REF-text	Debug-text
Phase 63		Procedure A1-text			Debug-text
Phase 64				REF-text	

Figure 61. Types of Compiler Text Produced by Each Phase

MICROFICHE DIRECTORIES

This section contains two directories to be used in conjunction with microfiche listings of the compiler. Microfiche names are usually the same as the load module names shown in the directories. Figure 62 associates load modules (listed in phase order) with object module names, the CSECTs they contain, the flowcharts in which they appear, and the chapters of this publication in which they are described.

Figure 63 associates external symbols (listed in alphabetical order) with the load modules in which they appear. Those external symbols that are CSECT names are designated as "SD." Those that are location definitions within a CSECT are designated as "LD."

Load Module	Entry Point	CSECT Names	Flow-charts	Refer to Chapter	Function
IKFCBL00	START	IKFCBL00 PHOSECT2 PHOTBST1 PHOTBST2 TBDATA	AA	Phase 00	Provides an interface between the operating phases of the compiler and between the system and the compiler. Handles tables and the dictionary for the operating phases, as well as requests for system input/output operations.
IKFCBL01		IKFCBL01	BA	Phase 01	Contains installation default values of compilation parameters.
IKFCBL02		IKFCBL02 IKFC21	BB	Phase 02	Initializes the compiler.
IKFCBL03	PH03	IKFCBL03	BC	Phase 03	Issues error messages and returns to phase 00 to terminate a compilation.
IKFCBL04	STRTPH04	IKFCBL04		Phase 04	Performs COPY/BASIS functions.
IKFCBL05		IKF0501		Phase 05	Analyzes syntax of source program and inserts syntactic markers for Lister processing.
IKFCBL06		IKF0601		Phase 06	Inserts cross-reference information into source program based on syntactic markers.
IKFCBL08		IKFC801		Phase 08	Produces Lister output.
IKFCBL10	PH1A	IKF101 . . . IKF118	CA-CD	Phase 10	Reads the Identification, Environment, and Data Divisions of the source program and stores the information in tables, the COMMON communications area, and Data IC-text for phase 20.
IKFCBL12	PHRW	IKF101 . . IKF114	CE-CJ	Phase 12	Processes Report Section of the Data Division and generates Report Writer Subprogram.

Figure 62. Load Module Directory (Part 1 of 4)

Load Module	Entry Point	CSECT Names	Flow-charts	Refer to Chapter	Function
IKFCBL1B	PH 1B	SDDEF1 IKF101 . . IKF10C	CK	Phase 1B	Reads the Procedure Division of the source program and stores its information in tables, the dictionary, and P0-text and Listing A-text.
IKFCBL2C	PH20	SDDEF2 (or IKF200) IKF201B IKF202 . . IKF209	DA-DD	Phase 20	Reads the Data IC-text from phase 10 and creates ATF-text for phase 22. Produces incomplete Data A-text for VALUE clauses.
IKFCBL22	START22	IKF202 . . IKF209	DE-DL	Phase 22	Reads Data IC-text and ATF-text and generates dictionary entries. Also generates Q-Routines and DEF-text, and completes Data A-text from phase 20.
IKFCBL21	PH2	IKF201 IKF203 IKF205 IKF207 IKF209 IKF211 IKF213 IKF214 IKF215	DM-DN	Phase 21	Completes FD and SD dictionary entries and writes Data A-text for DCBs, DECBs, and buffers.
IKFCBL25	PHASE25	IKF251 IKF252 IKF25A	DO-DR	Phase 25	Produces the DATATAB and OBODOTAB tables on the Debug data set (SYSUT5) if the SYNDMP option is in effect.
IKFCBL3C	IKFCBL3C	IKFCBL3C	EA-EC	Phase 3	Reads the P0-text from phase 1B, replaces the procedure-names with their dictionary attributes, expands SEARCH statements and CORRESPONDING clauses, and writes the text as P1-text for phase 4. Also produces a Data Division glossary, if requested. Writes procedure name DEF-text for phase 6A, if a cross-reference listing has been requested.
IKFCBL35	IKFCBL35	IKF40A IKF40B IKF40BL IKF40C . . IKF409 IKF401 IKF406		Phase 35	Scans USE FOR DEBUGGING declaratives. Inserts debug verbs into procedure IC-text, if required.

Figure 62. Load Module Directory (Part 2 of 4)

Load Module	Entry Point	CSECT Names	Flow-charts	Refer to Chapter	Function
IKFCBL40	PH4 (or CBLIKF40 or PHINIT)	IKF40A IKF40B IKF40B1 IKF40C IKF40D IKF40E IKF40F IKF409 IKF401 IKF406	FA-FC	Phase 4	Performs syntax analysis on the P1-text from phase 3, expands each complex or implied verb string into a series of simpler strings, thus producing P2-text for phases 50 and 51. Produces ATM-text for the UNSTRING verb.
IKFCBL45	PHASE45 (or PH45)	IKF450 . . IKF453	FD-FF	Phase 45	Translates ATM-text from phase 4 for the UNSTRING verb into P2-text for phase 51.
IKFCBL50	PHASE50	IKF501 . . IKF503 IKF50A . . IKF50F IKF504 IKF505	GA-GE	Phase 50	Begins to break down the P2-text from phase 4 into assembler-language-like statements that generally have a one-to-one correspondence to machine instructions. These are written as Intermediate A-text, which consists of Intermediate Procedure A-text and Intermediate Optimization A-text for input to phase 51. Phase 50 also produces Optimization A-text to help phase 6 or 62 to eliminate storage duplication.
IKFCBL51	PHASE51	IKF501 IKF503 IKF502 IKF50A IKF50G . . IKF50M IKF504 IKF505	GF-GJ	Phase 51	Completes the breaking down of P2-text into assembler-language-like statements that generally have a one-to-one correspondence to machine instructions. These are written as Final Procedure A-text for phase 6 or phases 62 and 63. It also produces Optimization A-text to help phase 6 or 62 to eliminate storage duplication.
IKFCBL60	PHASE6 (or PH6)	IKF601 . . IKF609 IKF610	HA-HF	Phase 6	Combines all the information from Procedure A-text, Optimization A-text, and Data A-text into an object module. Produces an object program listing, if requested. Writes REF-text on reference to data-names, file-names, and procedure-names for phase 6A, if a cross-reference listing has been requested. Writes Debug-text if the STATE option has been requested.
IKFCBL62	PHASE62	IKF62 IKF623 IKF625 IKF626 IKF627 IKF628	IA-ID	Phase 62	Reads Optimization A-text and optimizes literals and virtuals; reads Procedure A-text and calculates Procedure block numbers. Produces partial listings for DMAP, CLIST, and PMAP.

Figure 62. Load Module Directory (Part 3 of 4)

Load Module	Entry Point	CSECT Names	Flow-charts	Refer to Chapter	Function
IKFCBL63	PHASE63	IKF63 IKF631	IE-IJ	Phase 63	Produces Procedure A1-text from Procedure A-text, generating all remaining instructions for the object program with the exception of certain load instructions. Writes Debug-text if the STATE or SYMDMP option has been requested.
IKFCBL64	PHASE64	IKF64 IKF643 IKF644 IKF645 IKF6455 IKF646 IKF647 IKF648	IK-IO	Phase 64	Processes Data A-text and Procedure A1-text and completes the object text. Produces an object program listing, if requested. Writes REF-text for phase 6A.
IKFCBL65	PHASE65 (or PH65)	IKF651 IKF652 IKF653	IP-IQ	Phase 65	Produces debugging information for the SYMDMP, FLOW, and/or STATE options in the TGT. Adds the PROCTAB and SEGINDX tables to the object module after INIT3 for STATE; completes the Debug data set (SYSUT5) for SYMDMP. Called only if SYMDMP, FLOW, and/or STATE has been requested.
IKFCBL6A		IKF6A01 IKF6A02 IKF601A	IR	Phase 6A	Writes a cross-reference listing from DEF-text and REF-text. listing has been requested. listing has been requested.
IKFCBL70	PH7 (or IKFCBL70)	IKF701 IKF702 IKF703	JA	Phase 70	Writes error messages for phases 10 through 65, if a listing of error messages has been requested. Called only if program errors have been detected during compilation.
IKFCBL71	IKFCBL71	IKF711	JA	Phase 71	Contains message text for error messages generated by phases 20, 22, 21, or 25.
IKFCBL72	IKFCBL72	IKF721	JA	Phase 72	Contains message text for error messages generated by phases 3, 4, 45, 50, 51, 6, 62, 63, 64, or 65.
IKFCBL80	IKFCBL80	CHKCOPY CHKGLBLS DATASCAN ENVSCAN FIPSVT GETLINE GETWORD IDSCAN IKFCBL80 MSGHNDLR MSGWRITE PROSCAN PUTLINE VERBCHK		Phase 80	Scans the source program for deviations from the Federal Information Processing Standard (FIPS) and produces a listing.

Figure 62. Load Module Directory (Part 4 of 4)



External Symbol*	Type	Load/ Object Module	CSECT where LD Appears
CBLIKF40 (or PH4 or PHINIT)	LD	IKFCBL40	IKF408
CHKCOPY CHKGLBLS DATASCAN ENVSCAN FIPSVT GETLINE GETWORD IDSCAN	SD	IKFCBL80	
IKFCBL00	SD	IKFCBL00	
IKFCBL01	SD	IKFCBL01	
IKFCBL02	SD	IKFCBL02	
IKFCBL03	SD	IKFCBL03	
IKFCBL04	SD	IKFCBL04	
IKFCBL20	SD	IKFCBL20	
IKFCBL22	SD	IKFCBL22	
IKFCBL30	LD	IKFCBL30	
IKFCBL60	SD	IKFCBL60	
IKFCBL65	SD	IKFCBL65	
IKFCBL71	SD	IKFCBL71	
IKFCBL72	SD	IKFCBL72	
IKFCBL80	LD	IKFCBL80	IKFCBL80
IKF0501	SD	IKFCBL05	
IKF0601	SD	IKFCBL06	
IKF0801	SD	IKFCBL08	
IKF021	SD	IKFCBL02	
IKF101 . . .	SD . . .	IKFCBL10 . . .	
IKF118	SD	IKFCBL10	
*Use of "also" implies two separate CSECT cards, where one CSECT has a length of zero in storage; "or" applies to LDs which are equated.			

Figure 63 (Part 1 of 5). External Symbol Directory

External Symbol*	Type	Load/ Object Module	CSECT where LD Appears
IKF101 . . .	SD . . .	IKFCBL12 . . .	
IKF114	SD	IKFCBL12	
IKF101 . . .	SD . . .	IKFCBL1B . . .	
IKF109	SD	IKFCBL1B	
IKF201 IKF203 IKF205 IKF207 IKF209 IKF211 IKF213 IKF214 IKF215	SD . . . . . . . SD	IKFCBL21 . . . . . . . IKFCBL21	
IKF201B IKF202 IKF203 IKF204 IKF205 IKF206 IKF207 IKF208 IKF209	SD . . . . . . . SD	IKFCBL20 . . . . . . . IKFCBL20	
IKF202 . . .	SD . . .	IKFCBL22 . . .	
IKF209	SD	IKFCBL22	
IKF251 IKF252 IKF25A	SD SD SD	IKFCBL25	IKFI51
IKF40A IKF40B IKF40B1 IKF40C IKF40D IKF40E IKF40F IKF409 IKF401 IKF406	SD . . . . . . . . SD	IKFCBL40 . . . . . . . IKFCBL40	
*Use of "also" implies two separate CSECT cards, where one CSECT has a length of zero in storage; "or" applies to LDs which are equated.			

Figure 63 (Part 2 of 5). External Symbol Directory

External Symbol*	Type	Load/ Object Module	CSECT where LD Appears
IKF450	SD	IKFCBL45	
IKF451	.	.	
IKF452	.	.	
IKF453	SD	IKFCBL45	
IKF501	SD	IKFCBL50	
IKF502	.	.	
IKF503	.	.	
IKF50A	.	.	
IKF50B	.	.	
IKF50C	.	.	
IKF50D	.	.	
IKF50E	.	.	
IKF50F	.	.	
IKF504	.	.	
IKF505	SD	IKFCBL50	
IKF501	SD	IKFCBL51	
IKF502	.	.	
IKF503	.	.	
IKF50A	.	.	
IKF50G	.	.	
IKF50H	.	.	
IKF50I	.	.	
IKF50J	.	.	
IKF50K	.	.	
IKF50L	.	.	
IKF50M	.	.	
IKF504	.	.	
IKF505	SD	IKFCBL51	
IKF6A01	SD	IKFCBL6A	
IKF6A02	SD	IKFCBL6A	
ILF601	SD	IKFCBL60	
.	.	.	
.	.	.	
IKF609	SD	IKFCBL60	
IKF651	SD	IKFCBL65	
.	.	.	
.	.	.	
IKF656	SD	IKFCBL65	

\*Use of "also" implies two separate CSECT cards, where one CSECT has a length of zero in storage; "or" applies to LDs which are equated.

Figure 63 (Part 3 of 5). External Symbol Directory

External Symbol*	Type	Load/ Object Module	CSECT where LD Appears
IKF701	SD	IKFCBL70	
.	.	.	
.	.	.	
IKF703	SD	IKFCBL70	
IKF711	SD	IKFCBL71	
IKF721	SD	IKFCBL72	
MSGHNDLR	SD	IKFCBL80	
MSGWRITE	SD	IKFCBL80	
OPENEXIT	LD	IKFCBL80	IKFCBL80
PHASE45	LD	IKFCBL45	IKF450
PHASE50	LD	IKFCBL50	IKF504
PHASE51 (or PH51)	LD	IKFCBL51	IKF504
PHASE6 (or PH6)	LD	IKFCBL60	IKF601
PHASE65 (or PH65)	LD	IKFCBL65	IKF651
PHINIT (or PH4 or CBLIKF40)	LD	IKFCBL40	IKF408
PHRW	LD	IKFCBL12	IKF101
PHOSECT2	SD	IKFCBL00	
PH0TBST1	SD	IKFCBL00	
PH0TBST2	SD	IKFCBL00	
PH03	LD	IKFCBL03	IKFCBL03
PH1A	LD	IKFCBL10	IKF101
PH1B	LD	IKFCBL1B	IKF101

\*Use of "also" implies two separate CSECT cards, where one CSECT has a length of zero in storage; "or" applies to LDs which are equated.

Figure 63 (Part 4 of 5). External Symbol Directory

External Symbol*	Type	Load/ Object Module	CSECT where LD Appears
PH2	LD	IKFCBL21	IKF201
PH20	LD	IKFCBL20	IKFCBL20
PH4 (or CBLIKF40 or PHINIT)	LD	IKFCBL40	IKF408
PH45	LD	IKFCBL45	IKF450
PH6 (or PHASE6)	LD	IKFCBL60	IKF601
PH65 (or PHASE65)	LD	IKFCBL65	IKF651
PH7	LD	IKFCBL70	IKF701
PROCSCAN	SD	IKFCBL80	
PUTLINE	SD	IKFCBL80	
RTRN85	LD	IKFCBL80	IKFCBL80
SDDEF1 (also IKF101)	SD	IKFCBL10	
SDDEF1	SD	IKFCBL1B	
SDDEF2 (also IKF200)	SD	IKFCBL20	
START	LD	IKFCBL00	PHOSECT2
START22	LD	IKFCBL22	IKF202
TBDATA	SD	IKFCBL00	
VERBCHK	SD	IKFCBL80	
*Use of "also" implies two separate CSECT cards, where one CSECT has a length of zero in storage; "or" applies to LDs which are equated.			

Figure 63 (Part 5 of 5). External Symbol Directory

COMMUNICATIONS AREA (COMMON)

The communications area (COMMON) is resident in storage throughout compilation at the beginning of phase 00. The format of COMMON is defined as DSECTs in the remainder of the phases and, therefore, each phase can refer to any cell in COMMON by its own name.

Phase 00 passes a parameter list to each phase, the first word of which always contains the address of COMMON. The phases use this value to set up and maintain a register which points to COMMON, as shown in Figure 64.

Much of the information saved in COMMON by phases 10 through 51 is used by phase 6 or 62 to form the Task Global Table (TGT) and Program Global Table (PGT) of the object program.

When the BATCH option has been specified, certain cells in COMMON are reset to their original values by routines in phase 02 for subsequent compilations after the first. These cells are indicated by an asterisk following the cell name.

Phase	Register
01	**
02	10
03	12
04	**
05	09
06	12
08	09
10	1
12	1
1B	1
20	11*
22	1
21	1
25	10
3	**
35	**
4	**
50	10
51	10
6	9
62	9
63	9
64	9
65	9
6A	10
70	9
80	**

\*Except during ACCMET routine.  
 \*\*No register maintained throughout phase.

Figure 64. Registers Pointing to COMMON

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>
		<u>Hex.</u>	<u>Decimal</u>	
COS	12	0	0	Executable instructions. This section of code is used as an entry point to phase 00 by other phases. See "Receiving Control from Another Phase" in the chapter "Phase 00."
TIB0-TIB35*	8 each	00C	12	Table Information Blocks (TIBs) used by TAMER (see "Appendix A. Table and Dictionary Handling"). TIB20 is reserved for the DICOT table, and TIB30 is reserved for the HASH table. The rest are assigned to various compiler tables throughout compilation; one TIB may be reassigned when the table for which it was used is released.
APRIME AINSRT ADSTAT RELADD TAMNAD	4 each	12C 130 134 138 13C	300 304 308 312 316	Address constants of TAMER routines used by the phases in table management requests.
ACCESSW*	1	140	320	ACCESS initialization switch (see "Appendix A. Table and Dictionary Handling").
AMAINF	3	141	321	Pointer to the main free area for tables and the dictionary.
ALSTAM	4	144	324	Pointer to the last entry in the TAMM table (see "Appendix A. Table and Dictionary Handling").
LOCCTR*	4	148	328	Relative address of the next location available in the object program. It is incremented by phases 22 and 21 as they assign locations to data and then by phase 6 or phases 62, 63, and 64 as they assign locations to the Global Tables and procedure instructions.
PROGID*	8	14C	332	PROGRAM-ID from the Identification Division of the source program. It is saved to be used as the CSECT name of the object module. If the program is segmented, the name is the CSECT name of the root segment, and its first six characters are used with priority numbers to name the other segments.
LABELS*	2	154	340	Label information.
PRBLDISP	2	156	342	Displacement of PROCEDURE BLOCK cells in the PGT.
PNCTR*	2	158	344	Used in phase 1B as a counter for assigning unique PN numbers to source program procedure-names. In phase 6, it is set to the displacement of the PN field from the beginning of the PGT.
GNCTR*	2	15A	346	Used in phases 10, 1B, 22, 4, 50, and 51 as a counter for assigning unique GN numbers to compiler-generated procedure-names. In phase 6 or 62, it is set to the displacement of the GN field from the beginning of the PGT.
VIRCTR*	2	15C	348	Used in phases 50 and 51 as a counter for assigning unique identifying numbers to virtuals (names of external procedures). In phase 6 or 62, it is set to the displacement of the VIRTUAL field from the beginning of the PGT.

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>
		<u>Hex.</u>	<u>Decimal</u>	
LTLCTR*	2	15E	350	Used in phases 50 and 51 as a counter to save the number of literals. In phase 6 or 62, it is set to the displacement of the LITERAL field from the beginning of the PGT.
WCMAX*	2	160	352	Set by phases 50 and 51 to the size of the largest work area needed by any COBOL library subroutine. In phase 6 or 62, it is set to the displacement of the WORKING CELLS field from the beginning of the TGT.
TSMAX*	2	162	354	Set by phases 50 and 51 to the maximum number of doubleword cells needed for temporary storage at execution time by arithmetic verbs. In phase 6 or 62, it is set to the displacement of the TEMP STORAGE field from the beginning of the TGT.
TS2MAX*	2	164	356	Set by phases 50 and 51 to the number of bytes needed for temporary work areas by nonarithmetic statements. In phase 6 or 62, it is set to the displacement of the TEMP STORAGE-2 field from the beginning of the TGT.
ODOCTR*	2	166	358	Set in phase 22 to the number of Q-Routines generated to initialize a field which contains an OCCURS...DEPENDING ON clause the object of which is an item in Working-Storage or Communication Section or is in a basic file. A Q-Routine is a subroutine which, at execution time, calculates the length of a variable-length field created by the OCCURS...DEPENDING ON option, and the location of the variably-located field which may follow it. It is used in phase 6 or 64 to set up table QTBL.
CKPCTR*	2	168	360	Set in phase 21 to the number of checkpoint requests. It is used in phase 6 or 62 to allocate space for the CHECKPT CTR field of the TGT. Phase 6 or 62 sets it to the displacement of the CHECKPT CTR field from the beginning of the TGT.
SBLCTR*	2	16A	362	Used in phase 22 as a counter for assigning unique identifying numbers for secondary base locators (SBLs). In phase 6 or 62, it is set to the displacement of the SBL field from the beginning of the TGT.
VLCCTR*	2	16C	364	Used in phase 22 as a counter for assigning unique identifying numbers for variable length cells (VLCs). In phase 6 or 62, it is set to the displacement of the VLC field from the beginning of the TGT.
BLLCTR*	2	16E	366	Used in phase 22 to assign unique identifying numbers to Linkage Section base locators. In phase 6 or 62, it is set to the displacement of the BLL field from the beginning of the TGT.
SEQERR*	2	170	368	Count of source cards whose user-written card numbers are out of sequence. Set by phases 10 and 1B, and used by phase 70 in error message processing.
DICND2*	4	174	372	Dictionary pointer for the last dictionary entry made in phase 22.

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>												
		<u>Hex.</u>	<u>Decimal</u>													
DICND1*	4	178	376	Dictionary pointer for the last dictionary entry made in phase 1B.												
WSDEF*	7	17C	380	Set in phase 22 to the last seven bytes of the Data A-text element for Working-Storage Section address definition, which gives the first base locator number, the starting address, and the length of the Working-Storage Section. When phase 6 or 62 assigns permanent base registers for base locators, it uses this information because it assigns base registers to the Working-Storage Section first.												
				<table border="1"> <thead> <tr> <th><u>Name</u></th> <th><u>Bytes</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>WSSTRT</td> <td>3</td> <td>Starting address of Working-Storage</td> </tr> <tr> <td>WSBL</td> <td>1</td> <td>BL number assigned to the beginning of Working-Storage</td> </tr> <tr> <td>WSSIZE</td> <td>3</td> <td>Number of bytes occupied by Working-Storage</td> </tr> </tbody> </table>	<u>Name</u>	<u>Bytes</u>	<u>Meaning</u>	WSSTRT	3	Starting address of Working-Storage	WSBL	1	BL number assigned to the beginning of Working-Storage	WSSIZE	3	Number of bytes occupied by Working-Storage
<u>Name</u>	<u>Bytes</u>	<u>Meaning</u>														
WSSTRT	3	Starting address of Working-Storage														
WSBL	1	BL number assigned to the beginning of Working-Storage														
WSSIZE	3	Number of bytes occupied by Working-Storage														
ERRSEV	1	183	387	Set by phases 21, 3, 4, 50, and 51 to the highest error severity level encountered in any phase. When phase 00 returns to the operating system, it passes a return code which indicates the highest error severity level encountered in the program.												
DICADR*	4	184	388	ACCESS communication cell (see "Table and Dictionary Handling").												
DLSVAL*	4	188	392	ACCESS communication cell.												
DICPTR*	1	18C	396	ACCESS communication cell.												
DCPTR*	3	18D	397	ACCESS communication cell.												
RPTSAV*	2	190	400	Set by phase 10 if a Report Writer save area is needed at execution time. Used by phase 6 or 62 to determine whether that area should be set in the TGT and then set to the displacement of the RPTSAV field from the beginning of the TGT.												
SA2CTR*	2	192	402	Set in phase 51 if a USE AFTER (BEFORE) STANDARD LABEL or USE AFTER STANDARD ERROR procedure is encountered. If this field is set, phase 6 or 62 reserves an additional area (SAVE AREA-2) in the TGT and enters its displacement in this cell.												
LCSECT*	4	194	404	Length of the object module CSECT.												
RGNCTR*	2	198	408	Set by phase 6 to the number of unique GNs or, under the optimizer version of the compiler, set by phase 51 to the number of GNs requiring an address constant cell in the PGT.												
ERF4SW*	1	19A	410	Switch used by phases 6 or 62 and 64 and 70.												
PTYNO*	1	19B	411	Set by phase 51 to current priority number, and used by phase 00.												

Cell	No. of Bytes	Displacement		Purpose
		Hex.	Decimal	
COMMAD	2	19C	412	A comma followed by a decimal point. If the DECIMAL-POINT IS COMMA clause is specified, the order of the two is reversed; that is, a decimal point is followed by a comma. This cell is set by phase 10. This cell is used by phase 6 or 62 to set SWITCH in the TGT, or by phase 65 to set SWITCH with the STATE and/or FLOW options.
	2	19E	414	Unused.
AMOVDC*	4	1A0	416	Address constant for TAMER routines.
SDSIZ*	4	1A4	420	Set by phase 22 to the size of the largest Sort File Description (SD) Entry in the program.
SEGLMT*	1	1A8	424	Priority number of the highest-numbered Procedure Division section to be considered part of the root segment. Set in phase 10 to the value specified in the SEGMENT-LIMIT clause, or to 49. If phase 1B finds that the program is not segmented, it is set to hexadecimal 'FF' as an indication to later phases.
CURSGN*	1	1A9	425	Set in phase 10 to contain the literal specified in the CURRENCY-SIGN clause, and used by phase 20 to recognize this literal.
DATABDSP	2	1AA	426	Contains displacement into SYSUT5 block of first entry in DATATAB table, set by phase 25 for use by phase 65.
INDEX1*	2	1AC	428	Number of index-names defined in INDEXED BY clause. Set in phase 6 or 62 to the displacement of the INDEX field from the beginning of the TGT.
IOPTRCTR*	2	1AE	430	Number of input/output pointers for APPLY WRITE-ONLY clause.
TS3MAX*	2	1B0	432	Set by phases 50 and 51 to the number of bytes needed for temporary storage for the SYNCHRONIZED option.
TS4MAX*	2	1B2	434	Set by phase 50 to the number of bytes needed for temporary storage by table-handling verbs.
FLWSZ*	1	1B4	436	Set by phase 02 to the number of traces requested for the flow trace option. The default is 99. Used by phase 65 to fill in the DEBUG TABLE in the TGT.
SYSTX	1	135	437	To save SYSX from PARM field for a batch compile.
	1	1B5	437	Unused.
RPNCNTR	2	1B6	438	Under the optimizer version of the compiler, set by phase 51 to the number of PNs requiring an address constant cell in the PGT. Used by phases 62, 63, and 64.
AGETALL	4	1B8	440	Address constant for TAMER routines.
IDENTL*	4	1BC	444	Set in phase 6 or 64 to the relative location of the first executable instruction.
BLCTR*	2	1C0	446	Used in phase 22 as a counter for assigning unique identifying numbers to base locators for files and



<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>
		<u>Hex.</u>	<u>Decimal</u>	
				Working-Storage Section. In phase 6 or 62, it is set to the displacement of the BL field from the beginning of the TGT.
VNCTR*	2	1C2	450	Used in phase 4 as a counter for assigning unique identifying numbers to variable procedure-names. In phase 6 or 62, it is set to eight times the phase 4 value.
ONCTR*	2	1C4	452	Used in phase 51 as a counter to assign unique identifying numbers to ON control cells. In phase 6 or 62, it is set to the displacement of the ONCTL field from the beginning of the TGT.
PFMCTR*	2	1C6	454	Used in phase 4 as a counter to assign unique identifying numbers to PERFORM control cells. In phase 6 or 62, it is set to the displacement of the PFMCTL field from the beginning of the TGT.
PSVCTR*	2	1C8	456	Used in phase 4 as a counter to assign unique identifying numbers to PERFORM save cells. In phase 6 or 62, it is set to the displacement of field PFMSAV from the beginning of the TGT.
XSACTR*	2	1CA	458	Relative location within an EXHIBIT or SORT save area of the next area to be assigned. It is used by phase 51 in processing EXHIBIT or SORT statements and then is set to the total number of bytes needed for the save area. In phase 6 or 62, it is set to the displacement of field XSA from the beginning of the TGT.
				(Note: this counter is used and then incremented, unlike other counters which are incremented and then used. The increment is equal to the number of bytes in the save area used.)
XSWCTR*	2	1CC	460	Used by phase 51 as a counter to assign unique identifying numbers to EXHIBIT first-time switches and special ON switches. In phase 6 or 62, it is set to the displacement of field XSASW from the beginning of the TGT.
PH6ERR	2	1CE	462	Used by phases 6 or 62 and 64 and 65 to indicate that an error message is to be generated by phase 70. Bits 0-9 are correlated to messages IKF6001I - IKF6010I. Phase 70 checks bits 0-9 and if a bit is set to 1, the corresponding message is generated.
RELLOC*	4	1D0	464	Set in phase 6 or 62 to the relative location, within the object module or root segment, of the beginning of the TGT.
GTLNG*	2	1D4	468	Set in phase 6 or 62 to the length of the TGT.
VNILOC*	2	1D6	470	Set in phase 6 or 62 to the relative location of the VN field from the beginning of the PGT.
VNLOC*	2	1D8	472	Set in phase 6 or 62 to the relative location of the VN field from the beginning of the TGT.
SUBCTR*	2	1DA	474	Used in phase 4 as a counter to assign unique identifying numbers to subscripted references. In phase 6 or 62, it is set to the displacement of field SUBADR from the beginning of the TGT.

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>
		<u>Hex.</u>	<u>Decimal</u>	
PARMAX*	2	1DC	476	Set in phase 50 to the size of the parameter area needed for parameter lists for macro instruction expansion of some of the source statements. In phase 6 or 62, it is set to the displacement of the PARAM field from the beginning of the TGT.
PRBLNUM	1	1DE	478	Set by phase 62 to the number of Procedure blocks in the program; used by phases 63 and 64.
SPACING*	1	1DF	479	Set by phase 02 to the number of spaces between lines in the listing. Used by phase 6 or 64 for the statistics portion of the listing.
CORESIZE	4	1E0	480	Set by phase 02 to the total number of bytes available for this compilation. Used by phase 6 or 64 for the statistics portion of the listing.
COMFLOW	1	1E4	484	Value for FLOW from EXEC card.
	3	1E5	485	Unused
FIL5BUF	4	1E8	488	Used by phase 02 to store the address of the SYSUT5 buffer. Phases 25 and 65 also use this cell.
ADATAB	4	1EC	492	Note address for the first block of the DATATAB table of SYSUT5.
DATATBNM	2	1F0	496	Number of DATATAB blocks on SYSUT5.
OBODOTBN	2	1F2	498	Total number of bytes used for OBODOTAB entries on SYSUT5, including the slack bytes needed to align each OBODOTAB entry on a fullword boundary and unused bytes at the end of a block.
NODECTR	2	1F4	500	Number of node counters.
PROCCTR	2	1F6	502	Procedure-name counter.
AMICTR	2	1F8	504	Used by phase 21 as a counter for assigning unique identifying numbers for File Information Blocks. Phase 6 or 62 sets the field to the displacement of the FIB field from the beginning of the TGT.
DBGLOC	2	1FA	506	Displacement of TDBGTRA in the TGT.
SWITV2	1	1FC	508	Switch

<u>Name</u>	<u>Bit</u>	<u>Meaning</u>
NODUMP	1	The compiler will not issue an abend for D-level message condition and will generate the message. (NODUMP option)
CNTFDECL	0	Declaratives specified for COUNT.
COBOL2	2	ANS 1974 interpretation.
LSTRETXT	3	If E-text from phase 04.
INFOMSG	4	On if messages present that FIPS cannot process.

3 1FD 509 Unused

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>																					
		<u>Hex.</u>	<u>Decimal</u>																						
TMCNTBSZ	4	200	512	Size of count table.																					
VTINITVN	2	204	516	Virtual number for VSAM.																					
	2	206	518	Reserved																					
AMILOC	2	208	520	Set by phase 62 to the number of FIB cells. Tested by phase 64.																					
INTVIRT	2	20A	522	Virtual number for Initialization routines ILBOINT0. Used by phases 62-64.																					
LOCTNCTT	4	20C	524	Start of count table.																					
VIOVIRN	2	210	528	Virtual number for ILBOV2C0. Used by phases 62-64.																					
LISTERSW	1	212	530	Switch																					
<table border="1"> <thead> <tr> <th><u>Name</u></th> <th><u>Bit</u></th> <th><u>Option</u></th> </tr> </thead> <tbody> <tr> <td>LSTRDECK</td> <td>0</td> <td>FDECK</td> </tr> <tr> <td>LSTRPCH</td> <td>1</td> <td>CDECK</td> </tr> <tr> <td>LSTRCOMP</td> <td>2</td> <td>LSTCOMP</td> </tr> <tr> <td>LSTRONLY</td> <td>3</td> <td>LSTONLY</td> </tr> <tr> <td>LSTRPRC2</td> <td>4</td> <td>LCOL2</td> </tr> <tr> <td>LSTR132</td> <td>5</td> <td>L132</td> </tr> </tbody> </table>					<u>Name</u>	<u>Bit</u>	<u>Option</u>	LSTRDECK	0	FDECK	LSTRPCH	1	CDECK	LSTRCOMP	2	LSTCOMP	LSTRONLY	3	LSTONLY	LSTRPRC2	4	LCOL2	LSTR132	5	L132
<u>Name</u>	<u>Bit</u>	<u>Option</u>																							
LSTRDECK	0	FDECK																							
LSTRPCH	1	CDECK																							
LSTRCOMP	2	LSTCOMP																							
LSTRONLY	3	LSTONLY																							
LSTRPRC2	4	LCOL2																							
LSTR132	5	L132																							
COLLOVAL	1	213	531	Collating sequence LOW VALUE.																					
COLHIVAL	1	214	532	Collating sequence HIGH VALUE.																					
CDLCCTR	3	215	533	LOCCTR for CD INITIAL																					
COLLITNO	2	218	536	Literal number of PCS.																					
	2	21A	538	Unused.																					
DICTNAME	4	21C	540	Pointer to dictionary name from LATPTR.																					
LNGDSP	2	220	544	Displacement of first LINAGE-COUNTER.																					
LNGBL	2	222	546	BL number of first LINAGE-COUNTER.																					
APLSCALL	4	224	548	Address of PLSCALL routine in phase 00.																					
	1	228	552	Unused.																					
BUGSTCRD	3	229	553	Card number of first PN following declaratives.																					
	14	22C	556	Unused																					
INDEX	6	23A	570	Index all																					
DATE*	15	240	576	Set by phase 02 to the date and time of compilation; reused as follows by phases 10 and 1B:																					
<table border="1"> <thead> <tr> <th><u>Bytes</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>0-3</td> <td>Number of data-names</td> </tr> <tr> <td>4-7</td> <td>Number of verbs</td> </tr> <tr> <td>8-11</td> <td>Number of source cards</td> </tr> <tr> <td>12-14</td> <td>Unused</td> </tr> </tbody> </table>					<u>Bytes</u>	<u>Meaning</u>	0-3	Number of data-names	4-7	Number of verbs	8-11	Number of source cards	12-14	Unused											
<u>Bytes</u>	<u>Meaning</u>																								
0-3	Number of data-names																								
4-7	Number of verbs																								
8-11	Number of source cards																								
12-14	Unused																								
CRDNUM	3	24F	591	Used to hold object deck card number during processing by phases 62 through 64.																					
BCDISP	2	252	594	Set by phase 6 or 62 to the displacement of the EBCDIC NAME field in the PGT.																					

Licensed Material - Property of IBM

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>
		<u>Hex.</u>	<u>Decimal</u>	
BCDCTR	2	254	596	Set by phase 6 or 62 to the number of EBCDIC names to be placed in the PGT. This number is multiplied by 8 during phase 6 or 62 processing and the resulting amount of space is reserved in the PGT.
LINECNTX	2	256	598	To save LINECNT from PARM field for a batch compile.
COMPILE	4	258	600	Number of compilations for the BATCH option.
CNTLINE	2	25C	604	Set by phase 02 to the total number of lines per listing page. Used by phase 6 or 64 for the statistics portion of the listing.
ERRNUM*	2	25E	606	Number of errors for this compilation.
DBGODISP	2	260	608	Displacement of ILBODBG0 virtual cell from the beginning of the PGT.
SABCTR*	2	262	610	Set to four times the maximum number of files specified in any OPEN statement. Phase 6 or 62 reserves an additional area (SAVE AREA-3) in the TGT and enters its displacement in this cell.
VCONDISP	2	264	612	Displacement of VCON TBL field in TGT.
	2	266	614	Unused
SYNADR01	8	268	616	Used by phase 03 to store registers 0 and 1 for the SYNADAF macro instruction.

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>
		<u>Hex.</u>	<u>Decimal</u>	
ADDRCARD	4	270	624	Address of the CBL card saved by phase 02 or 10 for the BATCH option.
DCBCTR*	2	274	628	Used in phase 21 as a counter for assigning unique identifying numbers for DCBs (data control blocks). In phase 6 or 62, it is set to the displacement of the DCBADR field from the beginning of the PGT.
OPTINSW	1	276	630	Used to hold PHZSW until PHZSW is reset for the BATCH option.
OPTINSW1	1	277	631	Used to hold PHZSW1 until PHZSW1 is reset for the BATCH option.
OPTINSW2	1	278	632	Used to hold PHZSW2 until PHZSW2 is reset for the BATCH option.
OPTINSW3	1	279	633	Used to hold PHZSW3 until PHZSW3 is reset for the BATCH option.
OPTINSW4	1	27A	634	Used to hold PHZSW4 until PHZSW4 is reset for the BATCH option.
NUMINCR	1	27B	634	SYMDMP card number increment.
DEFCNT	4	27C	636	Number of definition text elements.
PHZSW	1	280	640	Set by phase 02 from the compilation options. If the bit is on, the option was chosen.

Equate

<u>Name</u>	<u>Bit</u>	<u>Option</u>
LIST	0	SOURCE
LISTX	1	PMAP
DECK	2	DECK
LINK	3	LOAD
SEQ	4	SEQ
FLAGW	5	FLAGW
LIBR	6	LIB
ZWB	7	ZWB

SYMSK

When SYNTAX is specified Phase 00 turns off all the bits to negate the options listed below. If CSYNTAX is specified and one or more error (E) or disaster (D) level message is found, phase 21, 30, 40, or 50 turns off all the bits to negate the options listed below.

LISTX	LINK
DECK	

PHZSW1 1 281 641 Same as PHZSW for additional options.

Equate

<u>Name</u>	<u>Bit</u>	<u>Option</u>
XREF	0	XREF
CLIST	1	CLIST
SYM	2	DMAP
FLOW	3	FLOW
SXREF	4	SXREF
APOST	5	APOST
MAPSP	6	SUPMAP
TRUNC	7	TRUNC

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u> <u>Hex. Decimal</u>		<u>Purpose</u>																																	
SYMSK1				Same as SYMSK for the following options:  <table border="0"> <tr> <td>FLOW</td> <td>TRUNC</td> </tr> <tr> <td>CLIST</td> <td>SYM</td> </tr> <tr> <td>SXREF</td> <td>XREF</td> </tr> </table>	FLOW	TRUNC	CLIST	SYM	SXREF	XREF																											
FLOW	TRUNC																																				
CLIST	SYM																																				
SXREF	XREF																																				
PHZSW2	1	282	642	Same as PHZSW for additional options.  <table border="0"> <tr> <td colspan="3">Equate</td> </tr> <tr> <td><u>Name</u></td> <td><u>Bit</u></td> <td><u>Option</u></td> </tr> <tr> <td>TERM</td> <td>0</td> <td>TERM</td> </tr> <tr> <td>NUM</td> <td>1</td> <td>NUM</td> </tr> <tr> <td>DYNAM</td> <td>2</td> <td>DYNAM</td> </tr> <tr> <td>BATCH</td> <td>3</td> <td>BATCH</td> </tr> <tr> <td>NAME</td> <td>4</td> <td>NAME</td> </tr> <tr> <td>SYMCAN</td> <td>5</td> <td>SYMDMP canceled</td> </tr> <tr> <td>STATE</td> <td>6</td> <td>STATE</td> </tr> <tr> <td>SYMDMP</td> <td>7</td> <td>SYMDMP</td> </tr> </table>	Equate			<u>Name</u>	<u>Bit</u>	<u>Option</u>	TERM	0	TERM	NUM	1	NUM	DYNAM	2	DYNAM	BATCH	3	BATCH	NAME	4	NAME	SYMCAN	5	SYMDMP canceled	STATE	6	STATE	SYMDMP	7	SYMDMP			
Equate																																					
<u>Name</u>	<u>Bit</u>	<u>Option</u>																																			
TERM	0	TERM																																			
NUM	1	NUM																																			
DYNAM	2	DYNAM																																			
BATCH	3	BATCH																																			
NAME	4	NAME																																			
SYMCAN	5	SYMDMP canceled																																			
STATE	6	STATE																																			
SYMDMP	7	SYMDMP																																			
SYMSK2				Same as SYMSK for the following options:  <table border="0"> <tr> <td>SYMDMP</td> <td>STATE</td> </tr> <tr> <td>NAME</td> <td></td> </tr> </table>	SYMDMP	STATE	NAME																														
SYMDMP	STATE																																				
NAME																																					
PHZSW3	1	283	643	Same as PHZSW for additional options.  <table border="0"> <tr> <td colspan="3">Equate</td> </tr> <tr> <td><u>Name</u></td> <td><u>Bit</u></td> <td><u>Option</u></td> </tr> <tr> <td>OPT</td> <td>0</td> <td>OPT</td> </tr> <tr> <td>RESEXEC</td> <td>1</td> <td>RESIDENT</td> </tr> <tr> <td>SYNTAX</td> <td>2</td> <td>SYNTAX</td> </tr> <tr> <td>CSYNTAX</td> <td>3</td> <td>CSYNTAX</td> </tr> <tr> <td>NOPRINT</td> <td>4</td> <td>(TSO only)</td> </tr> <tr> <td>TEST</td> <td>5</td> <td>TEST</td> </tr> <tr> <td>COBDBG</td> <td>5</td> <td>TEST</td> </tr> <tr> <td>ENDJOB</td> <td>6</td> <td>ENDJOB</td> </tr> <tr> <td>VERBR</td> <td>7</td> <td>VERB</td> </tr> </table> <p>The remaining bits are unused.</p>	Equate			<u>Name</u>	<u>Bit</u>	<u>Option</u>	OPT	0	OPT	RESEXEC	1	RESIDENT	SYNTAX	2	SYNTAX	CSYNTAX	3	CSYNTAX	NOPRINT	4	(TSO only)	TEST	5	TEST	COBDBG	5	TEST	ENDJOB	6	ENDJOB	VERBR	7	VERB
Equate																																					
<u>Name</u>	<u>Bit</u>	<u>Option</u>																																			
OPT	0	OPT																																			
RESEXEC	1	RESIDENT																																			
SYNTAX	2	SYNTAX																																			
CSYNTAX	3	CSYNTAX																																			
NOPRINT	4	(TSO only)																																			
TEST	5	TEST																																			
COBDBG	5	TEST																																			
ENDJOB	6	ENDJOB																																			
VERBR	7	VERB																																			
SYMSK3				Same as SYMSK for the following:  <table border="0"> <tr> <td>TEST</td> </tr> </table>	TEST																																
TEST																																					
PHZSW4	1	284	644	Same as PHZSW for additional options.  <table border="0"> <tr> <td colspan="3">Equate</td> </tr> <tr> <td><u>Name</u></td> <td><u>Bit</u></td> <td><u>Option</u></td> </tr> <tr> <td>VSUM</td> <td>0</td> <td>VBSUM</td> </tr> <tr> <td>VREF</td> <td>1</td> <td>VBREF</td> </tr> <tr> <td>COUNTL</td> <td>2</td> <td>COUNT</td> </tr> <tr> <td>TIMERL</td> <td>3</td> <td></td> </tr> <tr> <td>NEWADV</td> <td>4</td> <td>ADV</td> </tr> </table>	Equate			<u>Name</u>	<u>Bit</u>	<u>Option</u>	VSUM	0	VBSUM	VREF	1	VBREF	COUNTL	2	COUNT	TIMERL	3		NEWADV	4	ADV												
Equate																																					
<u>Name</u>	<u>Bit</u>	<u>Option</u>																																			
VSUM	0	VBSUM																																			
VREF	1	VBREF																																			
COUNTL	2	COUNT																																			
TIMERL	3																																				
NEWADV	4	ADV																																			
	1	285	645	Unused																																	
SWITCH*	2	286	646	TRACE, DEBUG, SYMDMP, and Q-Routine information.  <table border="0"> <tr> <td colspan="3">Equate</td> </tr> <tr> <td><u>Name</u></td> <td><u>Bit</u></td> <td><u>Meaning</u></td> </tr> <tr> <td>SWTRCE</td> <td>0</td> <td>Set by phase 1B if TRACE statement is encountered so that phase 4 will generate TRACE coding at each procedure-name definition.</td> </tr> </table>	Equate			<u>Name</u>	<u>Bit</u>	<u>Meaning</u>	SWTRCE	0	Set by phase 1B if TRACE statement is encountered so that phase 4 will generate TRACE coding at each procedure-name definition.																								
Equate																																					
<u>Name</u>	<u>Bit</u>	<u>Meaning</u>																																			
SWTRCE	0	Set by phase 1B if TRACE statement is encountered so that phase 4 will generate TRACE coding at each procedure-name definition.																																			

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement Hex. Decimal</u>	<u>Purpose</u>
NO6A	1		Set by phase 6 or 64 to indicate to phase 00 that the SXREF or XREF option has been canceled.
RPTWTR	2		Set by phase 10 to indicate whether to call phase 12.
SPILL	3		Set by phase 00 if dictionary spill occurs in phase 1B, 22, or 21.
COLATON	4		Switch indicating PROGRAM COLLATING SEQUENCE, not NATIVE, exists.
MQVAR	5		Set by phase 22 if it builds a QVAR table.
	6		Unused.
MQFILE	7		Set by phase 22 if it builds a QFILE table.
SYMIFP	8		Set by phase 25, when SYMDMP or TEST is requested, there is a floating-point item in the program. Tested by phase 62 to determine whether a virtual for ILBOTEF3 is to be generated.
SYS5TD	9		Set on if the SYSUT5 data set is on tape; set off if SYSUT5 is on disk.
SORTRTN	10		Set by phase 3 if the SORT-RETURN special register for the Sort Feature is specified and used by phase 51.
RERUNN	11		Set by phase 10 if the RERUN clause for the Sort Feature is specified, and used by phases 21, 51, and either 6 or 62 and 64.
DSOU	12		Source indicator for statistics.
NOFITSW	13		Indicates to phases 00 and 70 that phase 6 or 64 wrote E-text on SYSUT3.
DOPH7	14		Tested by phase 6 or 62 or 64 to determine whether to call phase 70.
RDERRFIL	15		Indicates that phase 70 must read E-text on SYSUT4.

PH1BYTE\* 1 288 648 Switch.

<u>Equate Name</u>	<u>Bit</u>	<u>Set by</u>	<u>Used by</u>
F3TEXT	0	3,51	6
QRTN1PBL	1	62	63
BASIS	2	02	00
F4TEXT	3	21,3,4,51	6 or 64
S370IN	4	02	50,51
INITBIT	5	10	20
EOPPH1	6	10	1B
UPSIBT	7	10	20

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>																											
		<u>Hex.</u>	<u>Decimal</u>																												
STAESW	1	289	649	STAE information for the BATCH option.																											
				<table border="1"> <thead> <tr> <th><u>Equate Name</u></th> <th><u>Bit</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>STAEON</td> <td>0</td> <td>STAE executed.</td> </tr> <tr> <td>STAEFAIL</td> <td>1</td> <td>STAE was not successful.</td> </tr> </tbody> </table> <p>The remaining bits are unused.</p>	<u>Equate Name</u>	<u>Bit</u>	<u>Meaning</u>	STAEON	0	STAE executed.	STAEFAIL	1	STAE was not successful.																		
<u>Equate Name</u>	<u>Bit</u>	<u>Meaning</u>																													
STAEON	0	STAE executed.																													
STAEFAIL	1	STAE was not successful.																													
BATCHSW	1	28A	650	BATCH option information.																											
				<table border="1"> <thead> <tr> <th><u>Equate Name</u></th> <th><u>Bit</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>EOFSYSIN</td> <td>0</td> <td>End of file on SYSIN.</td> </tr> <tr> <td>ENDFOUND</td> <td>1</td> <td>Delimiter has been read.</td> </tr> <tr> <td>BOMBED</td> <td>2</td> <td>Compiler tried to terminate.</td> </tr> <tr> <td>ENDIN1A</td> <td>3</td> <td>End of file reached by phase 10.</td> </tr> <tr> <td>CARDHELD</td> <td>4</td> <td>Card read by phase 02 is to be passed to phase 10.</td> </tr> <tr> <td>DONTSET</td> <td>5</td> <td>Bypass routine.</td> </tr> <tr> <td>NOTFIRST</td> <td>6</td> <td>Set after the first compile.</td> </tr> <tr> <td>LINISOFF</td> <td>7</td> <td>SYSLIN is not open.</td> </tr> </tbody> </table>	<u>Equate Name</u>	<u>Bit</u>	<u>Meaning</u>	EOFSYSIN	0	End of file on SYSIN.	ENDFOUND	1	Delimiter has been read.	BOMBED	2	Compiler tried to terminate.	ENDIN1A	3	End of file reached by phase 10.	CARDHELD	4	Card read by phase 02 is to be passed to phase 10.	DONTSET	5	Bypass routine.	NOTFIRST	6	Set after the first compile.	LINISOFF	7	SYSLIN is not open.
<u>Equate Name</u>	<u>Bit</u>	<u>Meaning</u>																													
EOFSYSIN	0	End of file on SYSIN.																													
ENDFOUND	1	Delimiter has been read.																													
BOMBED	2	Compiler tried to terminate.																													
ENDIN1A	3	End of file reached by phase 10.																													
CARDHELD	4	Card read by phase 02 is to be passed to phase 10.																													
DONTSET	5	Bypass routine.																													
NOTFIRST	6	Set after the first compile.																													
LINISOFF	7	SYSLIN is not open.																													
SPACEX	1	28B	651	To save SPACE from PARM field for a batch compile.																											
DECBCT*	2	28C	652	Used in phase 21 as a counter for assigning identifying numbers to DECBs. In phase 6 or 62, it is set to the displacement of the DECBADR field from the beginning of the TGT.																											
DCBNOXX	2	28E	654	Passes the DCB number for the optimizer phases.																											
BUFSIZE	4	290	656	Set by phase 02 to the total number of bytes to be used for buffers. Used by phase 6 or 64 for the statistics portion of the listing.																											
CURCRD*	3	294	660	The card number of the text item currently being processed.																											
SWITCH2	1	297	663	Switch.																											
				<table border="1"> <thead> <tr> <th><u>Equate Name</u></th> <th><u>Bit</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>OPTDISP</td> <td>0</td> <td>Set to 1 by phase 50 if ILBODSP0 is to be called; set to 0 if ILBODSS0 is to be called</td> </tr> <tr> <td>PH63NOPR</td> <td>1</td> <td>Set to 1 by phase 62 if phase 63 is to perform no processing but call phase 64</td> </tr> </tbody> </table>	<u>Equate Name</u>	<u>Bit</u>	<u>Meaning</u>	OPTDISP	0	Set to 1 by phase 50 if ILBODSP0 is to be called; set to 0 if ILBODSS0 is to be called	PH63NOPR	1	Set to 1 by phase 62 if phase 63 is to perform no processing but call phase 64																		
<u>Equate Name</u>	<u>Bit</u>	<u>Meaning</u>																													
OPTDISP	0	Set to 1 by phase 50 if ILBODSP0 is to be called; set to 0 if ILBODSS0 is to be called																													
PH63NOPR	1	Set to 1 by phase 62 if phase 63 is to perform no processing but call phase 64																													



<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u> <u>Hex. Decimal</u>		<u>Purpose</u>
EOJCALL				2 Set to 1 by phase 00 if it receives a call for end-of-job processing.
RERUNBT				3 If there is a RERUN clause, phase 10 sets this bit to 1 to indicate to phase 51 to move the address of the Checkpoint subroutine (ILBOCKP0) into the exit list.

The remaining bits are unused.

SWITCH1 1 298 664 Used for SYMDMP and UNSTRING communication.

<u>Equate Name</u>	<u>Bit</u>	<u>Meaning</u>
RENAMON	0	Phase 22 built RENAMTB table.
OCCTBON	1	Phase 22 built OCCTBL table.
PH45BIT	2	Set by phase 4 to indicate that phase 45 is to be called for the UNSTRING verb.
CVBBIT	3	Set if the ILBOCVB0 subroutine is needed.
NEDBIT	4	Set if the ILBONED0 subroutine is needed.
APPWRO	5	APPLY WRITE ONLY specified in source program.
ANEBIT	6	Set if the ILBOANE0 subroutine is needed.
ADRSYM	7	Set if the ILBOADRO subroutine is needed.

	3	299	665	Unused
KTRMNATE	4	29C	668	Pointer to error code for phase 03. (See the chapter "Phase 03" for a list of the codes and their meanings.)
RELSPACA	4	2A0	672	Address of RELSPACE (TAMER address constant).
KALOUT	4	2A4	676	A (LOUT) -- pointer to SYSOUT DCB address.
KKADS5	4	2A8	680	A (DS5) -- pointer to SYSUT5 DCB address. Phase 65 uses this address in opening and closing SYSUT5.
LINKCNT	2	2AC	684	Set to the phase in main storage; used by phases 00 and 03.
KKPHOSW	1	2AD	686	Switch

<u>Equate Name</u>	<u>Bit</u>	<u>Meaning</u>
KKPHORT	7	Phase 00 has issued a RETURN macro instruction

The remaining bits are unused.

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>
		<u>Hex.</u>	<u>Decimal</u>	
PHZERR	1	2AE	687	Switch for error message.
PRINTBUF	1	2AF	688	Printer carriage control character.
PROGSW	1	2B0	689	Switch for progress message.
KKPGR70	3	2B2	690	X'F6FA00' -- phase 70 purge constant; reset by phase 03.
	1	2B5	693	Unused.
INITSIZE	2	2B6	694	Size of INIT1 as it is always generated.
DICND3	4	2B8	696	Set by phase 22 to the dictionary pointer for the last dictionary entry in the File Section.
PH25SW	2	2BC	700	Number of blocks written on SYSUT5 by phase 25.
SYSTDD	1	2BE	702	Used by phases 02, 60 and 62 to determine the final alphanumeric character for DISPLAY SYSOUx DD name.
FIPLVL	1	2BF	703	Set by phase 02 to indicate use level of FIPS flagging that is to be done by phase 80.
	12	2C0	704	Reserved
AHEADER	4	2CC	716	ADCON for header routine in phase 00.
ESDID	2	2D0	720	Used by phase 62 to pass last ESD ID number to phase 64.
BGALLPN	2	2D2	722	PN for USE-DEBUG all procedures.
BGALLPRI	1	2D4	724	Priority for USE-DEBUG QALL procedure.
V2BUGSW	1	2D5	725	DEBUG switch.

<u>Equate Name</u>	<u>Bit</u>	<u>Meaning</u>
V2BUGON	0	With debugging mode
V2BUGDCL	1	Debugging sections
BGALLPRC	2	USE-DEBUG all procedures

BUGBLLNO	2	2D6	726	BLL for DEBUG
BUGVLCNO	2	2D8	728	VLC for DEBUG
MAXBGITM	2	2DA	730	Maximum size for DEBUG-ITEM.

IDBYTES	28	2DC	732	One byte for each phase of the compiler. Each phase dynamically stores its change level number in its corresponding byte.								
IDPH00	1	2DC	732									
IDPH01	1	2DD	733									
IDPH02	1	2DE	734									
IDPH03	1	2DF	735									
IDPH04	1	2E0	736									
IDPH05	1	2E1	737									
IDPH06	1	2E2	738									
IDPH08	1	2E3	739									
IDPH10	1	2E4	740									
IDPH12	1	2E5	741									
IDPH1B	1	2E6	742									
IDPH20	1	2E7	743									
IDPH22	1	2E8	744									
IDPH21	1	2E6	745									
IDPH25	1	2EA	746									
IDPH30	1	2EB	747									
IDPH35	1	2EC	748									
IDPH40	1	2ED	749									
IDPH45	1	2EE	750									
IDPH50	1	2EF	751									
IDPH51	1	2F0	752									
IDPH60	1	2F1	753									
IDPH62	1	2F2	754									
IDPH63	1	2F3	755									
IDPH64	1	2F4	756									
IDPH65	1	2F5	757									
IDPH6A	1	2F6	758									
IDPH70	1	2F7	759									
IDPH71	1	2F8	760									
IDPH72	1	2F6	761									
IDPH80	1	2FA	762									
	12	2FB	763	Unused								
	1	307	775	Unused								
OUTLRECL	2	308	776	121 or 131 set by phase 02 for LISTER.								
OPTLVL	1	30A	778	LVL option saved for batch								
OPTSHV2	1	30B	779	Option in SWITV2 saved for batch								
OPTLSTR	1	30C	780	Option in LISTERSW saved for batch								
PHOSW	1	30D	781	Switch byte used in phase 0's.								
				<table border="0"> <thead> <tr> <th>Equate</th> <th>Name</th> <th>Bit</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td></td> <td>SYSLIBNO</td> <td>8</td> <td>Switch for SYSLIB not available.</td> </tr> </tbody> </table>	Equate	Name	Bit	Meaning		SYSLIBNO	8	Switch for SYSLIB not available.
Equate	Name	Bit	Meaning									
	SYSLIBNO	8	Switch for SYSLIB not available.									
	2	30E	782	Unused								
ALIBEOD	4	310	784	Address of EOD exit for library								
ALIBSYNA	4	314	788	Address of SYNAD exit for library.								
LIBBUF	4	318	792	Address of library input area (phase 04 to phase 00).								

COMPILER TABLE FORMATS

This chapter contains the formats of all the tables that are handled through the TAMER routines (see "Appendix A: Table and Dictionary Handling"). These tables are distinguishable from others in that (1) additional storage can be obtained for them, and (2) they can be left in TAMER table space by one phase to be used by another.

All other tables occupy fixed amounts of storage in the phases that use them. They are described in the chapters on the individual phases.

NOTES ON COMPILER TABLE FORMATS

- The top row of figures shows the number of bytes in the field.
- **Boxes that are shaded** define optional fields or a series of similar fields.
- c = the number of bytes in the field that follows.
- n = the total number of bytes that follow in the remainder of the entry.
- 1b = this field is one byte long.

ALPHTBL  
(TIB 27)

Purpose

Phase 10 builds this table from scan of alphabets in SPECIAL-NAMES paragraph.

1	Variable	1	256
C	alphabet-name	Switch	256-byte Translation Table
①		②	

Phases Involved

Phase 10 checks the table for alphabet-names found in CODE-SET clause. At end of SPECIAL-NAMES, phase 10 scans the table for alphabet-names found in PROGRAM COLLATING SEQUENCE.

Phase 1B checks the table for alphabet-names found in SORT or MERGE statements.

Phase 51 deletes this table at termination of processing.

① The count of the number of bytes in the name that follows.

Code	Meaning
01	PROGRAM COLLATING SEQUENCE
04	STANDARD-1 (ASCII), no 256-byte translation table needed.
08	NATIVE (EBCDIC), no 256-byte translation table needed.
80	Literal already defined.

If none of the above switches are on, the table entry is for a user-defined alphabet which is not the Program Collating Sequence. If the Literal Already Defined bit is on, the next two bytes contain the literal number (if not ASCII or EBCDIC).

BLASGTBL  
(TIB 16)

Purpose

Assign object-time permanently loaded registers.

1	1
Type cell	BL, BLL, or OVERFLOW number
①	

Entry Frequency

One entry for each register: 6-10.

Phases Involved

Phase 62 builds this table using the BLUSTBL table.

Phases 63 and 64 use the information to determine which BL or BLL or OVERFLOW cell is in a permanent register.

① The type cell contains one of the following values:

Code	Meaning
FF	TGT OVERFLOW
F0	PGT OVERFLOW
00	data BL
01	data BLL

OVERFLOW cells for the TGT and PGT are assigned registers first. Since the number of PROCEDURE BLOCK ADDRESS cells has not yet been determined, it is impossible to know if another OVERFLOW cell will be required for the PGT. Therefore, phase 62 assigns registers 6 - 9 to the known OVERFLOW cells and to the most used data BLs and BLLs and reserves register 10 for the possible PGT OVERFLOW cell. If no OVERFLOW cell is needed, register 10 is assigned to the next most used data BL or BLL.

BLUSTBL  
(TIB 10)

Purpose  
Contains a count of the references to each BL and BLL.

3
Usage counter

Entry Frequency  
One entry for each BL and BLL assigned to the Data Division.

Phases Involved  
Phases 50 and 51 build this table during the scan of P2-text. Phase 62 uses this table to assign registers to the most used data BLs/BLLs. BLs 1 through n are followed by BLLs 1 through m.

BLVNTBL  
(TIB 23)

Purpose  
Optimize generation of instructions to return control from a performed procedure to the GN return point.

2	2	1
GN number	VN number	Block number

Entry Frequency  
One entry for each EXIT statement in the range of a PERFORM statement.

Phases Involved  
Phase 62 builds this table during Optimization A-text processing upon reading a GN element for a PERFORM verb (24). It fills in the block number during Procedure A-text processing upon reading the VN reference element which follows the C004 element at the PERFORM statement exit. Phase 63 uses this table to determine whether the GN return point (C005 element) is in the same block as the EXIT statement and thus which Procedure Block number is contained in register 11 upon return from the performed procedure.

INDX  
3 11)

Purpose

Store information about the first card number for each program fragment and user-written discontinuity within a segment for use by the COBOL library subroutines when SYMDMP is specified.

3	1	1
Card/verb number for first card in this group	Priority	Relative fragment number within this priority
①		

Entry Frequency

One entry for each program fragment and one entry for each noncontiguous section other than the first within a segment.

Phases Involved

Phase 65 builds this table while reading Debug-text on SYSUT4 on building the PROCTAB table.

Phase 65 writes this table on SYSUT5 and COBOL library subroutines use this table to relate card numbers to entries in the PROCTAB table.

Bits

Contents

1-19 Card number  
20-23 Verb number  
(verb number is always 0 or 1)

TBL

3 8)

Purpose

Save RERUN statement information from source program scan.

Entry Frequency

One entry for each RERUN statement.

Phase Involved

Phase 10 builds this table from RERUN statement in the source program.

Phase 21 adds DCB number and uses this table to build RUNTBL and BSAM checkpoint file DCB.

8	1	1	2	4	2
External-name of checkpoint file	Type ①	Unused	DCB number of check-point file	"Integer" file	Chain pointer (contains zeros if no other entry for file)
				(contains SORT RERUN)	

Bit

Meaning, if on

0 Rerun every N record  
1 Rerun on END of REEL/UNIT  
2-7 Unused

CNTLTBL

①

Purpose  
Sorts data-names and procedure-names for the SXREF option.

4	2	2
Pointer to associated DATA record	Pointer to CONTROL record for lower name on first compare	Pointer to CONTROL record for lower name on next compare

Entry Frequency  
One entry (CONTROL record) for each DEF-text element.

Phases Involved  
Phase 6A builds and uses this table to reorder data-names and procedure-names alphabetically for the SXREF option.

- ① There is no TIB for this table. Phase 6A uses the phase 00 routine GETALL to get space, but moves data in and out of the table by itself.

CONDIS  
(TIB 14)

Purpose  
Store DISPLAY literals during literal optimization.

Variable
Literal

Entry Frequency  
One entry for each unique DISPLAY literal.

Phases Involved  
Phase 6 or, under the optimizer version of the compiler, phase 62 builds this table while processing Optimization A-text.  
Phase 6 or Phase 62 uses this table with CONTBL and LTLTBL to eliminate duplicate DISPLAY literals.

CONTBL  
(TIB 9)

Purpose  
Store each non-DISPLAY literal value during optimization of literals.

Variable
Literal

Entry Frequency  
One entry for each unique non-DISPLAY literal.

Phases Involved  
Phase 6 or, under the optimizer version of the compiler, phase 62 builds this table while processing optimization A-text.  
Phase 6 of phase 62 uses this table with CONDIS and LTLTBL to eliminate duplicate non-DISPLAY literals.



BL (Report Writer)

14) Purpose

Store information on control-names to check validity and build routines using them.

Entry Frequency

One for each control-name.

2	7	7	1	1	2
n	Duplicate name (-nnnn) in EBCDIC	Save name (-nnnn) in EBCDIC	Flag byte	Level of this control	GN number for control heading
	①	①	②		

Phases Involved

Phase 12 routine RDSCAN builds this table.

Phase 12 routine GNSPRT and most other routines use this table to create CTB-ROUT, SAV-ROUT, RET-ROUT routines.

2	2	2	Variable
GN number for control footing	Unused	Size of previous entry	Control-name including qualifiers, and subscripts (if any) in P0-text form.
		③	

TB

12) Purpose

Store virtual definition elements during virtual optimization.

Entry Frequency

One entry for each unique virtual.

8
Virtual

Phases Involved

Phase 6 or, under the optimizer version of the compiler, phase 62 makes an entry when it finds a virtual definition during Optimization A-text processing.

Phase 6 or phase 62 uses this table with VIRPTR table to eliminate duplicate references to virtuals.

In P0-text form:

Bytes	Contents
0	23
1	05
2	- (hyphen)
3-6	nnnn

- ② Bit      Meaning, if on  
 0-2      Unused  
 3      Control footing specified  
 4-6      Unused  
 7      Control heading specified
- ③ Size of previous entry = X'0001' for first entry in the table.

DATATBL

①

Purpose  
Store information for XREF or SXREF processing

3	33	2
Pointer to dictionary entry for data name or PN number	External name in EBCDIC, defining card number, and a variable number of referencing card numbers	Ascending source order pointer (SXREF only)

Entry Frequency  
One entry (DATA record) for each DEF-text element.

Phases Involved  
Phase 6A builds and uses this table in producing an XREF or SXREF listing.

2	1	3	1	3
Descending source order (SXREF only)	Offset in bytes from start of record for the next referencing card number	Pointer to current (last) OVERFLOW record	Length of external name	Pointer to first OVERFLOW record

①

Type will be 38 or 3C code for DATA A-text.

DATATBL

①

Purpose  
38 or 3C data A-text. Save elements until end of phase, when all DCBs have been generated and all addresses are known.

1	3	1
type	address where BL or BLL address will be stored	BL or BLL number

Entry Frequency  
One entry for each BL or BLL address elemented created.

Phases Involved  
Phase 21 builds this table for each 38 or 3C element it will be generating. Phase 21 uses this table to generate all 38 or 3C elements at end of phases.

①

Type will be 38 or 3C code for DATA A-text.

DBGTBL  
(TIB 13)

Purpose  
Store information on procedure-names referred to in DEBUG statements.

2	2
PN number for procedure-name	GN number for debug procedure associated with procedure-name

Entry Frequency  
One entry for each procedure-name referred to by DEBUG.

Phases Involved  
Phase 4 builds this table from PNs in P1-text referred to in DEBUG statements and from GNs from GNCTR in COMMON. Phase 4 uses this table to issue P2-text CALL statements to debug procedures.

There is no TIB for this table.  
 Phase 6A uses the phase 00 routine  
 GETALL to get space, but moves data  
 in and out of the table by itself.

T  
 06)

Purpose  
 Save newly created  
 Debug-text elements.

1	Variable
Number of bytes following	DEBUG verb string element

Phases Involved  
 Phase 35 builds this  
 table and uses it to  
 accumulate DEBUG verb text  
 while processing an input  
 verb string.  
 Phase 35 deletes this  
 table upon completion  
 of processing.

S  
 18)

Purpose  
 Store subscript-defining  
 string until all  
 subscripts in statement  
 are collected.

Variable		Variable
First element in string		Last element in string

Entry Frequency  
 One entry for current  
 string being built.

Phases Involved  
 Phase 4 builds this table  
 from P1-text of subscripted  
 data-name.  
 Phase 4 uses this table  
 with STRING table to issue  
 P2-text subscript strings.

3L (Report Writer)

17)

Purpose  
 Store information on  
 detail report group for  
 processing SUM...UPON  
 clauses and generating  
 detail-names.

30	1	2	1	2
Detail report group data-name		GN number for this detail report group		Displacement of entry in RWRTBL for report-name associated with this detail group
①		②		③

Entry Frequency  
 One entry for each  
 detail report group.

Phases Involved  
 Phase 12 builds this  
 table from scan of  
 01-level statements.  
 Phase 12 uses this table  
 to process SUM...UPON  
 clauses, and to generate  
 USM-ROUT routine.  
 Phase 1B uses this table  
 to generate detail-names.

2
GN number for USM-ROUT

④

- ① Left-justified, padded with binary zeros in low-order bytes.
- ② Length of preceding detail report group data-name.

- ③ Code for correlating SOURCE and SUM...UPON clauses.

Code	Meaning
00	This entry was made as the result of a detail report group encountered.
FF	This entry was made when an UPON clause was encountered. (This code is changed to 00 when a detail report group is encountered for the data-name.)

- ④ First entry in the table is a dummy.

DICOT  
(TIB 20)

Purpose  
Store starting address of each section in the dictionary.

1	3	8
①	Displacement of section in dictionary	②

Entry Frequency  
One entry for each dictionary Section.

Phases Involved  
Phases 1B and 22 build this table as they build the dictionary.  
Phases 1B, 22, 21, 25, and 30 use this table to find dictionary sections.

- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>① Bits</li> <li>0</li> <li>1</li> <li>2</li> <li>3</li> <li>4-7</li> </ul> | <p><u>Meaning, If ON</u></p> <ul style="list-style-type: none"> <li>Section is not in storage.</li> <li>Section is now in storage In interlude before phase 30 (21 or 25) if both bits 1 and 2 are on, the 3 bit is set on.</li> <li>Section has been spilled (note: 0 or 1 bit is on)</li> <li>This bit is set on and used only during phase 30 processing. A section, which has been spilled and read back into storage, has been modified and the copy on the external device is obsolete.</li> <li>Not used.</li> </ul> |
|---|---|

- ② Address on external device where section has been spilled.

DRPLTBL  
(TIB 25)

Purpose  
Store information for addressing BL, BLL, SBL, SBS, or BDISP address increment items.

1 bit	1 bit
0 = Load instruction required	0 = Register 14
1 = No load instruction required	1 = Register 15

Entry Frequency  
One entry for each of the above items if it is not assigned a permanent register.

Phases Involved  
Phase 62 builds this table during Procedure A-text processing.  
Phase 63 uses this table.

If a load instruction is to be generated, it generates the instruction and inserts the proper temporary register to address the item. If no load instruction is to be generated, it uses the proper temporary register as the base in the instruction.

DRPTBL  
(TIB 24)

Purpose  
Optimize the use of temporary registers 14 and 15.

1	1
Item type	Item number
①	

Entry Frequency  
One entry for each BL, BLL, SBL, SBS, or BDISP address increment if it is not assigned a permanent register and if a temporary register is unavailable.

Phases Involved  
Phase 62 builds this table and keeps the entries until a decision is made as to which temporary register, 14 or 15, should be used.

- |   |                                  |                         |
|---|----------------------------------|-------------------------|
| ① | <u>Code</u>                      | <u>Meaning</u>          |
|   | 80                               | BL                      |
|   | 40                               | BLL                     |
|   | 20                               | SBL                     |
|   | 10                               | SBS                     |
|   | 08                               | BDISP address increment |
|   | (code values are in hexadecimal) |                         |

DTAB  
(TIB 04)

Purpose  
Each entry describes a USE FOR DEBUGGING (UFD) operand.

1	1	1	2
Switch	PN	Priority	PN number
①	Reference element	number for PN	for USE

Phases Involved  
Phase 30 builds this table and makes entries for all the PN definitions in the program after the debug declaratives if the BGALLPRC switch is on. Phase 35 deletes this table upon completion of processing.

3	Variable
DICTPRT or PN number	Alpha Literal
②	

- |   |            |                              |
|---|------------|------------------------------|
| ① | <u>Bit</u> | <u>Meaning</u>               |
|   | 0          | File name                    |
|   | 1          | Procedure name               |
|   | 2          | CD name                      |
|   | 3          | Identifier                   |
|   | 4          | All references of identifier |

② The PN number is preceded by FF .

ENVTBL  
(TIB 3)

Purpose

Store file information from Environment Division to be merged with Data Division information to form Data IC-text.

3	8	2	1	1
Source card number	ddname portion of system-name of file padded with blanks, if necessary	Flag field	Flag field for VSAM	Buffer offset
		①	⑦	

Entry Frequency

One entry for each file.

Phases Involved

Phase 10 builds this table from Environment Division.

Phase 10 uses this table to merge with Data Division information.

2	1	1
Pointer to entry in PIOTBL	"Integer" from RESERVE AREA clause (contains zeros if no reserve area specified)	Unused
②		

2	2	2	1	2
Number of TRACK LIMIT tracks	Unused	Pointer to entry in CKPTBL	Number assigned to SAME AREA clause	Flag field
		②		④

1	1	5	1	2	1
⑤	Number assigned to SAME RECORD AREA clause	Unused	Ct of IDs for file	Disp. of IDs in INDTBL	Unused

1	1	2	1
Reserved	Ct of Password for ALT	Unused	Ct
⑧			

31	4	4	4	4
File-name in FD entry	TRACK AREA size	NOMINAL KEY and qualifiers	ACTUAL KEY and qualifiers	RECORD KEY and qualifiers
	⑥	⑥	⑥	⑥

4	4	4	4
Pointer to data-name qualifiers from option (REORG-CRITERIA)	FILE-STATUS and qualifiers	PASSWORD and qualifiers	RELATI KEY and qualifiers
⑤	⑥	⑥	⑥

① Bit	Meaning
0	1 = RANDOM ACCESS
1-3	Organization 000 = Not specified 001 = INDEXED 010 = DIRECT 'W' 011 = DIRECT 'D' 100 = RELATIVE
4-6	Unused
7	1 = Integer specified in RESERVE ALTERNATE AREA field 0 = No clause specified
8	'OPTIONAL' qualifier specified in SELECT
9	1 = SAME AREA specified
10	Unused
11	SAME RECORD AREA specified
12	Unused
13	1 = CKPTBL pointer appears in this entry
14	1 = PIOTBL pointer appears in this entry
15	Word 'ALTERNATE' specified in RESERVE clause

② Displacement for the first of two allowable entries for a file-name in table.

④ Bit	Meaning
0-1	TRACK-AREA 00 = Not specified 01 = Data-name 10 = Integer
2	1 = RELATIVE KEY
3	1 = NOMINAL KEY
4	1 = ACTUAL KEY
5	1 = RECORD KEY
6	1 = WRITE ONLY

7	FILE STATUS clause specified
8	1 = WRITE VERIFY
9-14	Unused (10=RESERVE integer invalid) (12 multiple files)
15	1 = RECORD OVERFLOW

⑤ Bits	Contents
0-3	Temporary storage for phases 10 and 1B
4	CORE-INDEX
5	1 = REORG-CRITERIA (APPLY)
6	ASCII
7	Unused

⑥ Bytes	Contents
0-1	Length of name(s) in bytes (or literal if TRACK AREA size is an integer)
2-3	Displacement of name in QNM TBL

⑦ Bit	Meaning
0	1 = ADDRESSED SEQUENTIAL
1	1 = Organization parameter omitted in system-name
2-4	ORGANIZATION clause
2	1 = RELATIVE
3	1 = INDEXED
4	1 = SEQUENTIAL
5	1 = Access Mode is Dynamic
6	ALTERNATE RECORD KEY specified
7	1 = Password data name with RECORD KEY clause.

⑧ This field is moved to FSTEXT set-up area and is used there to contain LINAGE information. It will not contain LINAGE information in the ENVTBL, and if used for any other purpose will be overlaid by FSTEXT.

**ERRTBL**  
(TIB 10)

Purpose  
Store E-text to separate it from Data A-text for phase 7.

Entry Frequency  
One entry for each message to be generated.

8	1	1	Variable	1	1	Variable
E-text for basic message	00	c	First message parameter	00	c	Last message parameter

Phases Involved  
Phase 6 or, under the optimizer version of the compiler, phase 64 builds this table from E-text interspersed with Data A-text. Phase 70 uses this table to generate error messages.

**FDTAB**  
(TIB 28)

Purpose  
Pass record description information from phase 22 to 21.

Entry Frequency  
One entry for each FD in source program.

2	2	1	1	2	1	3
Maximum record length	Minimum record length	First base locator number	Flag byte	Maximum label record size	Buffer offset	Dictionary pointer

Phases Involved  
Phase 22 builds from record descriptions in ATF-text. Phase 21 uses to generate DCBs, DECBS, and buffers

- ① **Bits**    **Use**
- 0-3    Number of base locators
- 4    ODO2 switch: ON, if any record descriptions contained more than one ODO clause.
- 5    ODO switch: ON, if any record description contained an ODO clause.
- 6    ODO object switch: ON, if any record description contained the object of an ODO clause.
- 7    SAME RECORE AREA for this SD
- ②    Used by phase 21 to get dictionary attributes when LATRNM returns a duplicate code.

**FNTBL**  
(TIB 10)

Purpose  
Store Environment Division information about a file for Procedure Division processing.

Entry Frequency  
One entry for each file.

2	2	2	2	2	2
Pointer to PIOTBL entry	Pointer to GVNMTBL	Unused	GN number for STANDARD ERROR	GN number for header labels	GN number for tra labels

Phases Involved  
Phase 10 builds this table from ENVTBL and Data Division. Phase 1B uses this table in Procedure Division processing.

2	2	1	1	Variable
GN number for EOVL labels	GN number for BOVL labels	Switch byte	c	File-name in EBCDIC



① Bits	Meaning, if on
0	ACCESS RANDOM
1	Mass storage file
2	LABEL RECORDS ARE STANDARD
3	LABEL RECORDS ARE OMITTED
4	BEFORE (in USE statement)
5	AFTER (in USE statement)
6-7	Unused

GCNTBL  
(TIB 24)

Purpose

Store card numbers for statements that contain NEXT GROUP or LINE clauses that may be in error; also, store card numbers for TYPE IS PAGE HEADING or TYPE IS PAGE FOOTING groups that may be in error.

3	3
①	Card Number

Entry Frequency

One entry for each clause in error.

Phases Involved

Phase 12 builds and uses. It makes an entry for each clause that conflicts with the PAGE LIMIT clause. Entries are saved until the end of the report, when it can be established whether these statements are actually in error, as signalled by the presence of at least one relative LINE or relative NEXT GROUP clause.

- ① These three bytes will contain the address of one of the following messages:

- MSG94, for a NEXT GROUP clause error
- MSG119, for a LINE clause error
- MSG165, for an illegal PAGE HEADING
- MSG166, for an illegal PAGE FOOTING

GNTABL  
(TIB 8)

Purpose

Determine which GNs require an address constant cell in the PGT.

2
GN number

Entry Frequency

One entry for each GN requiring an address constant cell in the PGT.

Phases Involved

Phase 62 builds this table from GNUREF elements in Optimization A-text.

Phase 63 uses this table to determine whether a GN requires an address constant cell.  
Phase 64 uses this table when determining the address in the PGT of the GN cell to be used in an instruction.

GNCALTBL  
 (TIB 16)

Purpose  
 Store GN numbers for Q-Routines.

1	2
Count-1 of Q-Routines to call	First GN number for Q-Routine call

Entry Frequency  
 One entry for each GN number.

Phases Involved  
Phase 51 builds this table and uses it to generate calls to the Q-routines after return from the CALL statement.

GNFWDBTB  
 (TIB 21)

Purpose  
 Optimize size of a procedure block.

2	1
GN number	Counter

Entry Frequency  
 One entry for each forward reference to a GN within a Procedure Block.

Phases Involved  
Phase 62 builds this table from Procedure A-text.  
Phase 62 uses this table to keep count of the number of 4-byte load instructions of the Procedure Block which might be needed if a new block is begun before the GN is defined.

GNLABTBL  
 (TIB 19)

Purpose  
 Determine inter-block and intra-block references.

1
Block number for GN

Entry Frequency  
 One entry for each GN.

Phases Involved  
Phase 62 enters in this table the block number for each GN as it reads Procedure A-text.  
Phase 63 extracts the block number in which a GN is defined each time a GN is referred to.

TBL

27)

Purpose

Determine displacements from the beginning of the block for GN definitions.

12 bits
Displacement from beginning of block for GN

Entry Frequency

One entry for each GN.

Phases Involved

Phase 63 builds this table during Procedure A-text processing.

Phase 64 uses this table to insert the displacement in generated instructions which address the GN.

8)

Purpose

Create and store a list of optimized GN numbers.

2
Number relative to beginning of PN cells

Entry Frequency

One entry for each GN number.

Phases Involved

Phase 6 builds this table from GNCTR and PN and GN equate strings.

Phase 6 uses this table to optimize procedure-names and process Procedure A-text and Listing A-text.

PK

10)

Purpose

Count length of group item while subordinate items are being processed.

4	2	2	3
Flag bytes	OD2TBL displacement of entry for OCCURS	Maximum number of occurrences	Source card number
	DEPENDENT ON object within group item	(0, if none)	
①	(0, if none)		

Entry Frequency

One entry for each group item being currently processed.

Phases Involved

Phase 22 builds this table from dictionary. Phase 22 uses this table to determine group item length.

3	3	4
Pointer to dictionary entry for REDEFINES object within group item (0, if none)	Pointer to dictionary entry for item	Maximum length of variable group item

1	3	1	1
Level number	Address parameters (idk)	Number of index-names (0, if none)	Number of keys (0, if none)

2	2	2
Displacement of entry for item in INDKEY table (0, if none)	Displacement of entry for item in SRCHKY table (0, if none)	Displacement entry for VAL clause literal in VALGRP tab if used (0, if not)

2	1
Displacement of entry for VALUE clause literal in VALTRU table, if used (0, if not)	Flag byte (2)

① Bits	Meaning
0	1 = Group occurs more than once; alignment required.
1	1 = Group contains object of OCCURS DEPENDING ON
2	1 = SYNC clause in item under group item
3	1 = SYNC clause in group item
4	1 = VALUE clause in group item
5	1 = Condition-name under group item
6	1 = Group is or is in a label record
7	1 = Item itself contains an OCCURS...DEPENDING ON clause
8-11	Minor code (see "LD ENTRY" in "Section 5. Data Areas Dictionary Entry Formats")
12-13	Number of subscripts required 00 = none 01 = 1 10 = 2 10 = 2 11 = 3
14	1 = Item contains an OCCURS or OCCURS...DEPENDING ON clause
15-31	Length of group or VLC

② Bits	Contents
0	1 = Master of an OCCURS... DEPENDING ON clause
1-3	Unused
4-7	Code    Meaning
	1000    Justified
	1111    Usage other than displ

**GVFNTBL  
(TIB 4)**

Purpose

Store displacement to entries in FNTBL for VSAM files referred to in USE AFTER STANDARD ERROR/EXCEPTION with GIVING option.

2
Displacement in FNTBL

Entry Frequency

One entry for each file-name mentioned in USE Declarative.

Phases Involved

Phase 1B builds the table using the FNTBL.

Phase 1B uses the table to locate the correct FNTBL for each filename mentioned in the Declarative and insert into FNTBL the displacement into the GVNMTBL of the fully qualified dataname in the GIVING option.

**GVNMTBL  
(TIB 3)**

Purpose

Store the data-name specified in the GIVING option of the STANDARD ERROR/EXCEPTION PROCEDURE Declarative for VSAM files.

1	Variable	1
00	data-name	Number of bytes in preceding field

For qualified data-names the following fields are added.

Entry Frequency

One entry for each Declarative. If the data-name is qualified the entry contains all of its qualifiers.

Variable	1	Variable	1
First qualifying data-name	Number of bytes in preceding field	Last qualifying data-name	Number of bytes in preceding field

Phases Involved

Phase 1B builds the table from the entries in the CURBCD and CURN data areas and for qualified data names from the QLTABL table.

Phase 1B uses the table.

**HASH**  
(TIB 30)

Purpose  
Store dictionary pointer for latest hash value.

3
---

Entry Frequency  
One entry for each hash value. The table is 521 bytes long, allowing for 177 possible hash values.

Dictionary
pointer or
zeros

Phases Involved  
Phase 10 builds this table.  
Phases 1B, 22, 21, 25, and 3 use this table.

**INDTBL**  
(TIB 4)

Purpose  
Store information about ALTERNATE KEYS and their passwords and/or duplicate status; used to construct IC-text elements for alternate keys.

4	4	1
RECORD	PASSWORD	FLAG
KEY		
(4)	(4)	(5)

Entry Frequency  
One entry for each ALTERNATE KEY clause.

Phases Involved  
Phase 10 builds INDTBL, then deletes it after Data IC-text is passed.

3
Generated Source
card number

(4) RECORD KEY is a two-byte length of the name, followed by a two-byte displacement of the name in QNMTBL. PASSWORD is a two-byte length of the name, followed by a two-byte displacement of the name in QNMTBL. The PASSWORD field is filled with zeros if no PASSWORD is specified.

(5) Flag  
Byte

<u>Meaning</u>
= 1 if another entry for the same file follows.
= 1 if PASSWORD is specified for this key.
= 1 if WITH DUPLICATES is specified.

**INDKEY**  
(TIB 31)

Purpose  
Store OCCURS DEPENDING ON information for use in table handling.

1	3	1	3	1
n	Dictionary	Flag	Dictionary pointer	Number of
	pointer to	byte	to object of OCCURS	index-names
	subject of		or maximum number	
table	(1)	(2)	of occurrences	(1)(3)

Entry Frequency  
One entry for each item that has an OCCURS clause and an INDEXED BY clause.

Phases Involved

Phase 22 builds this table from Data IC-text data-names with OCCURS and INDEXED BY clauses. Phase 3 uses this table to process SEARCH and SEARCH ALL verbs.

3		3	1
Dictionary pointer to first index-name ①		Dictionary pointer to last index-name ①	Number of keys

3	3
Dictionary pointer to first key ①	Dictionary pointer to last key ①

Contents of dictionary pointer:

Bits	Contents
0-1	Zeros
2-14	Dictionary section
15-23	Displacement in section

② Bit	Meaning
0-2	Unused
3	1 = Next three bytes contain dictionary pointer 0 = Next three bytes contain maximum number of occurrences
4	Unused
5	0 = Object of OCCURS...DEPENDING ON undefined 1 = No error found in OCCURS... DEPENDING ON object
6	Unused
7	1 = Error detected in key processing 0 = No error found

③ Field contains zeros if bit 5 in preceding field is set to 1.

2TBL  
B 17)

Purpose  
Store information on VSAM files for ORGANIZATION IS INDEXED.

3	3	1
Dictionary pointer for key ①	idk parameters for key ①	Level number

Entry Frequency

One entry for each FD entry for VSAM files with ORGANIZATION IS INDEXED.

Phases Involved

Phase 21 builds this table from Data IC-text. Phase 30 uses the table to build P1-text.

RECORD KEY, ALTERNATE RECORD KEY or RELATIVE KEY.

XTB  
B 34)

Purpose  
Save all index-names associated with a data item.

1	1	Variable
04 (hex)	c	Index-name in EBCDIC

Entry Frequency

One entry for each index-name associated with the data item currently being processed.

Phases Involved

Phase 10 builds this table from level number entries in the source program Data Division.

Phase 10 uses this table to append index-names to the Data IC-text LD entry for the data item.

**KEYTAB  
(TIB 26)**

Purpose

Save all key-names associated with a data item.

1	1	Variable
Flag	c	Key-name in EBCDIC
byte		
①		

Entry Frequency

One entry for each key-name associated with the data-item currently being processed.

Phases Involved

Phase 10 builds from level-number entries in source program Data Division.

Phase 10 uses this table to append key-name to Data IC-text LD entry for data item.

- ① Bits
- 0-5
- 6
- 7

Meaning, if on

None; contains zeros  
Descending key  
Ascending key

**KEYTBL  
(TIB 20)**

Purpose

Used in SEARCH verb processing to check whether keys in WHEN clause are valid for table in which binary search is being made.

3	1
Addressing parameters for item (IDK) from dictionary entry ①	②

Entry Frequency

One entry for each key in SEARCH ALL statement currently being processed.

Phases Involved

Phase 4 builds this table while processing a SEARCH ALL statement.

Phase 4 uses this table to make sure SEARCH ALL statement has correct keys for table and that all preceding keys are tested if the key is tested.



① Bits	Field	Meaning
0-3	i	Type of BL containing base address of area
		0000 = BL
		0001 = BLL
		0100 = SBL
4-15	d	Displacement from base address
16-23	k	BL number

② Initially 0, but set to 1 whenever a WHEN condition for KEY is found during processing of this SEAFCH ALL statement.

LABTBL  
(TIB 13)

Purpose

Save label-record data-names referred to in a Data IC-text FD entry.

2	1	Variable
Unused  c  LABEL RECOFD data-name in EBCDIC		

Entry Frequency

One entry for each LABEL RECORD data-name referred to in the Data IC-text FD entry currently being processed.

Phases Involved

Phase 20 builds this table from Data IC-text FD entries.

Phase 20 uses this table to differentiate label records from nonlabel records in processing Data IC-text LD entries.

LTLTBL  
(TIB 4)

Purpose

Contains pointers to CONTBL and CONDIS tables during optimization of literals.

2
Displacement from start of appropriate table of entry for literal

Entry Frequency

One entry for each reference to a literal.

Phases Involved

Phase 6 or, under the optimizer version of the compiler, phase 62 builds this table while building CONDIS and CONTBL.

Phase 6 or, under the optimizer version of the compiler, phase 64 uses this table with CONDIS and CONTBL to eliminate duplicate DISPLAY and non-DISPLAY literals.

MASTODO  
(TIB 13)

Purpose

Identify masters of OCCURS...DEPENDING ON clause if SYMDMP or TEST is specified.

3
Dictionary pointer for master of an ODO

Entry Frequency

One entry for each master of an OCCURS...DEPENDING ON clause.

Phases Involved

Phase 22 builds this table as it encounters OCCURS...DEPENDING ON clauses.

Phase 25 uses this table to identify master of OCCURS...DEPENDING ON clauses for the DATATAB table.

NPTTBL (Report Writer)

(TIB 18) Purpose

Store N.nnnn names that contain number of lines a particular report group occupies.

1	1	6
23 (hex)	07	Name in EBCDIC

Entry Frequency

One entry for each report group that contains PLUS clause. This table is cleared at the end of each RD.

Phases Involved

Phase 12 builds from relative line clauses. Phase 12 uses this table to generate Data IC-text LD entries at the end of the RD.

OBJSUB

(TIB 5)

Purpose

Relate files and CD entries, to objects and subjects of OCCURS...DEPENDING ON clauses. It is used to build the QFILE table.

2	2	2	2
DCB or CD Number	①	①	X'FFFF'

Entry Frequency

One entry for each file or CD entry whose record descriptions contain at least one object and/or subject of an OCCURS...DEPENDING ON clause.

Phases Involved

Phase 22 builds and uses this table to build the QFILE table.

① When present, this field contains the OD2TBL table displacement if the field refers to the object of an ODO clause, or the GN number of the subject if the field refers to the subject of an ODO clause. If the field contains a GN number, the high-order bit is ON.

CTBL  
IB 2)

Purpose

Store information about items in OCCURS and OCCURS...DEPENDING ON clauses if SYMDMP or TEST is specified and program contains an OCCURS or OCCURS...DEPENDING ON clause.

3	2
Dictionary pointer for subject of clause	Maximum number of occurrences ①

Entry Frequency

One entry for each subject of an OCCURS or of an OCCURS...DEPENDING ON clause.

2	1	2
Number of bytes to next occurrence ②	ODO Switch ③	Reserved for OBODOTAB pointer for object of ODO ④

Phases Involved

Phase 22 builds this table as it encounters an OCCURS or an OCCURS...DEPENDING ON clause.

Phase 25 uses this table and the QRTN and QITBL tables to build the ODOTBL table. The ODOTBL table is then used to fill in the OBODOTAB pointers in this table.

- ① The field contains either the number of occurrences for an OCCURS clause or the maximum number of occurrences for an OCCURS...DEPENDING ON clause.
- ② If the subject of the clause is a variable-length group, the field contains its VLC number.
- ③ If this byte contains 0, the entry is for an OCCURS clause; if it contains 1, the entry is for an OCCURS...DEPENDING ON clause.
- ④ The field is present only when the entry is for an OCCURS...DEPENDING ON clause. Phase 22 fills the field with zeros. Phase 25 enters the OBODOTAB pointer; then the contents of the field are as follows:

Bits	Contents
0-8	Relative block number within the OBODOTAB table
9-15	Displacement in fullwords within the block

ODOTBL  
IB 14)

Purpose

Determine which entries in the dictionary are objects of OCCURS...DEPENDING ON clauses and therefore must be entered in the OBODOTAB table if SYMDMP or TEST is specified and the program contains an OCCURS...DEPENDING ON clause.

3	2	2
Dictionary pointer for object of ODO	Displacement within OCCTBL for OBODOTAB pointer	OBODOTAB pointer for object of ODO

Entry Frequency

One entry for each OCCURS...DEPENDING ON clause.

Phases Involved

Phase 25 builds this table using the OCCTBL, QRTN, and QITBL tables.

Phase 25 uses this table to build the OBODOTAB table and to fill in the OBODOTAB pointers for objects of OCCURS...DEPENDING ON clauses in the OCCTBL table.

OD2TBL  
(TIB 9)

Purpose

Store objects of OCCURS DEPENDING ON clauses and their qualifiers for Q-Routine generation.

2	1	Variable	1
n	c	EBCDIC name of lowest-level qualifier	c

Entry Frequency

One entry for each OCCURS DEPENDING ON clause.

Phases Involved

Phase 10 enters EBCDIC names from OCCURS DEPENDING ON clauses.

Phase 22 uses this table to generate Q-Routines.

Variable	1	Variable	1
EBCDIC name of lowest-level qualifier	c	EBCDIC qualified name	00

OFLOTBL

①

Purpose

Store referencing card numbers when the DATATBL table entry is full. Used in processing for the SXREF or the XREF option

3	3	3	3	1	3
Referencing card number	Referencing card number	Referencing card number	Referencing card number	Off-set	Pointer to preceding OVERFLOW record

Entry Frequency

One entry for each referencing card number which cannot be stored in the DATATBL table entry for the data-name or the procedure-name.

Phases Involved

Phase 6A builds and uses this table to store referencing card numbers for the SXREF or the XREF option.

PFMTBL  
(TIB 12)

Purpose

Store procedure-names and VNs to be equated in PERFORM statements.

2	2
PN number of next procedure-name after end-of-range	VN number corresponding to PN that is the end-of-range

Entry Frequency

One entry for each delimiting procedure-name.

Phases Involved

Phase 4 builds this table from P1-text procedure-names and VNCTR in COMMON.

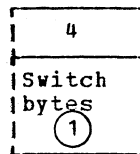
Phase 4 uses this table to keep track of delimiters of performed procedures to set up return VNs.

- ① There is no TIB for this table. Phase 6A uses the phase 00 routine GETALL to get space, but moves data in and out of the table by itself.
- ② This byte contains binary values 3, 6, 9, 12 as the OVERFLOW record contains 1, 2, 3, 4 referencing card numbers.

PIOTBL  
(TIB 7)

Purpose

Store input/output information for a file from Procedure Division.



Entry Frequency

One entry for each file.

Phases Involved

Phase 10 reserves space for one entry for each file.

Phases 1B and 12 complete entries from Procedure Division.

Phase 21 uses this table to generate Data A-text.

① Bit	<u>Statement referring to file, if bit is on</u>
0	OPEN INPUT
1	OPEN OUTPUT
2	OPEN I-O
3	CLOSE UNIT NO REWIND (OS)
4	WRITE AFTER ADVANCING
5	CLOSE WITH LOCK
6	CLOSE
7	REWRITE
8	RERUN
9	OPEN INPUT REVERSED
10	READ (SEEK IN DOS)
11	WRITE BEFORE ADVANCING
12	USING
13	GIVING
14	USE
15	WRITE AFTER POSITIONING
16	BEFORE in USE
17	OPEN NO REWIND
18	WRITE
19	USE ON file-name
20	START
21	INVALID KEY
22	REVERSED
23	RESERVED
24	KEY OF REFERENCE (READ and START)
25	DELETE
26	GIVING data-name for USE procedures
27	OPEN EXTEND
28	SEQUENTIAL ACCESS (DYNAMIC mode)
29	RESERVED
30	UNUSED
31	RESERVED

PNATBL  
(TIB 7)

Purpose  
Determine which PNs require an address constant cell in the PGT. 

3
PN number

Entry Frequency  
One entry for each PN requiring an address constant cell in the PGT.

Phases Involved  
Phase 62 builds this table from PNUREF elements in Optimization A-text.  
Phase 63 uses this table to determine which PNs require address constant cells.  
Phase 64 uses this table when determining the address in the PGT of the PN cell to be used in an instruction.

PNFWDBTB  
(TIB 20)

Purpose  
Optimize size of a Procedure Block. 

2	1
PN number	Counter

Entry Frequency  
One entry for each forward reference to a PN within a Procedure Block.

Phases Involved  
Phase 62 builds this table from Procedure A-text.  
Phase 62 uses this table to keep count of the number of 4-byte load instructions of the Procedure Block which might be needed if a new block is begun before the PN is defined.

PNLABTBL  
(TIB 18)

Purpose  
Determine interblock and intrablock references. 

1
Block number for PN

Entry Frequency  
One entry for each PN.

Phases Involved  
Phase 62 enters in this table the block number for each PN as it reads Procedure A-text.  
Phase 63 extracts the block number in which a PN is defined each time a PN is referred to.

PNLBDBL  
(TIB 26)

Purpose  
Determine displacements from the beginning of the block for PN definitions.

12 bits
Displacement from beginning of block for PN

Entry Frequency  
One entry for each PN.

Phases Involved  
Phase 63 builds this table during Procedure A-text processing.  
Phase 64 uses this table to insert the displacement in generated instructions which address the PN.

PNOUNT  
(TIB 14)

Purpose  
Stack operands of COMPUTE and IF statements.

Variable	1
Same operand as in P1-text, including byte 0 identification	①

Entry Frequency  
One entry for each operand in statement.

Phases Involved  
Phase 4 builds this table from P1-text scan.  
Phase 4 uses this table with PSIGNT table to stack operands until the string is ready to be created.

① Number of preceding bytes in this entry.

PNQTBLL  
(TIB 6)

Purpose  
Store information on references to qualified PNs for completion of dictionary entry.

1	1	Variable	2	1	1
n	c	Procedure-name in EBCDIC	Flag bytes	Unused	22
			①		

Entry Frequency  
One entry for each qualified PN.

Phases Involved  
Phase 1B builds this table from Procedure Division.  
Phase 1B uses this table to complete procedure-name dictionary entries.

1	Variable
c	Procedure-name qualifier in EBCDIC

PNTABL  
(TIB 5)

Purpose

Store information on references to each unqualified PN for later completion of PN's dictionary entry.

1	1	Variable	2	1
n	c	Procedure- name in EBCDIC	Flag bytes	Dictionary search code
			①	②

Entry Frequency

One entry for each PN that is not qualified.

Phases Involved

Phase 1B builds this table from Procedure Division.

Phase 1E uses this table to complete procedure-name dictionary entries.

① Bit	Meaning, if on
0	Procedure-name
1	Section-name
2	Either name follows THRU in PERFORM...THRU, or follow PERFORM without THRU
3	Referred to by ALTER
4	Procedure-name of GO TO
5	Procedure-name of EXIT
6	Procedure-name following TO PROCEED TO in an ALTER statement
7	Unused
8	Referred to in DEBUG
9	Defined in DEBUG
10	Dummy section name
11	Defined in Declaratives Section (or in DEBUG statement referring to such section). Bits 12-15 describe the type of section.
12	USE...ERROR section-name
13	USE...LABELS section-name
14	Unused
15	USE...REPORTING section-name

② Bit	Meaning, if on
0	Dictionary was search for this PN before this section.
1-7	Unused.

PNTBL  
(TIB 7)

Purpose

Create and store list of optimized PN numbers.

2
Displacement of PN from start of PN cells in object module

Entry Frequency

One entry for each optimized PN number.

Phases Involved

Phase 6 builds this table from PNCTR in COMMON and PN equate strings.

Phase 6 uses this table to optimize PNs and process Procedure A-text.



BL  
6)

Purpose  
Optimize procedure-names by eliminating those not referred to or, under the optimizer version of the compiler, determine entry points.

1 bit
1 = Name is referred to
0 = Name is not referred to

Entry Frequency  
One entry for each source program procedure-name.

Phases Involved

Phase 51 reserves space for as many entries as there have been PNs counted in PNCTR. These entries are initially set to 0. In later processing, phase 51 turns on a bit each time the PN associated with the bit is referred to.

Phase 6 uses this table to eliminate names not referred to or, under the optimizer version of the compiler, phase 62 uses this table to determine which PNs are entry points.

CINDX  
B 5)

Purpose  
Store information about the PROCTAB table which is written on SYSUT5 if the SYMDMP option is in effect.

3	3	4
Card/verb number for first entry in PROCTAB block ①	Relative address of code for this entry within segment	Device address of PROCTAB block

Entry Frequency  
One entry for each block of PROCTAB entries.

Phases Involved

Phase 65 builds this table while reading Debug-text on SYSUT4 and building the PROCTAB table.

Phase 65 writes this table on SYSUT5 and COBOL library subroutines use this table to address entries in the PROCTAB table.

Bits    Contents  
0-19    Card number  
20-23    Verb number

PSHTBL  
(TIB 17)

Purpose  
Store GN numbers of  
'ELSE' branches in nested  
and compound IF state-  
ments.

1	2
16 = Master GN	GN number
0 = Otherwise	

Entry Frequency  
One entry for each false  
branch.

Phases Involved  
Phase 4 builds this table  
from P1-text and GNCTR  
in COMMON.  
Phase 4 uses this table  
to generate branches.

PSIGNT  
(TIB 15)

Purpose  
Store operators in  
COMPUTE and IF  
statements.

2
P1-text code for sign

Entry Frequency  
One entry for each  
operator in the state-  
ment being processed.

Phases Involved  
Phase 4 builds this table  
from P1-text.  
Phase 4 stacks operators  
until strings are ready  
to generate. This table  
is used with PNOUNT.

PTRFLS  
(TIB 16)

Purpose  
Store branches in an IF  
statement.

1	2	2
1 = 'NOT' is active	GN number for fall-through bypass	GN number for branch
0 = 'NOT' is not active		

Entry Frequency  
One entry for each  
decision in statement.

Phases Involved  
Phase 4 builds this table  
from P1-text compound IF  
statements and GNCTR in  
COMMON.  
Phase 4 uses this table  
with PSHTBL to select true  
and false branches for  
P1-text.

BTBL  
IB 2)

Purpose

Pass information from phases 10 and 12 to phase 1B.

Entry Frequency

One entry made at end of phase 10 processing and, optionally, at the of phase 12 processing.

Phases Involved

Phases 10 and 12 build this table from stored information. Phase 1B and/or 12 moves data into its own data areas.

2	8	8	3	80
n	Card number for last record read (packed)	Card number for last record read (unpacked)	Unused	Last record read

18	6	82	6	2
Error information on record	Sequence number of last record read	INSERT card work area from BASIS	Sequence number from BASIS	Phase switches (1)

24	82	8	8	2
Sequence numbers specified on BASIS card	Work area for BASIS card	Latest COPY library-name	BASIS library-name	Phase switches (2)

3	4	80	8	2	1
Current card number	Phase switches (3)	Double buffer for COPY... REPLACING	Library-name for double-buffer COPY	Contents of 72 and 73 columns	Phase switches (4)

Bits	Byte 1	Byte 2
0	Unused	INDLSW
1	COPYSW	INSTSW
2	BASISW	INDERR
3	Unused	INOWSW
4	Unused	DELSW1
5	Unused	DELSW2
6	Unused	DELSW3
7	CPYXSW	CPYCSW

(3) Bits	Byte 1	Byte 2	Byte 3	Byte 4
0	Unused	CDSURP	CON2SW	Unused
1	Unused	BUF2SW	NWCDSW	SURPSW
2	Unused	BUF3SW	SKCDSW	BUF5SW
3	Unused	BUF4SW	Unused	Unused
4	REPSW	CONTSW	Unused	Unused
5	Unused	Unused	Unused	Unused
6	Unused	Unused	Unused	Unused
7	Unused	Unused	DBRDSW	Unused

Bits	Byte 1	Byte 2
0	Unused	TOTUSD
1	Unused	Unused
2	Unused	Unused
3	Unused	USEPDL
4	Unused	USEPDE
5	Unused	SRTSW
6	Unused	Unused
7	RPTWSW	Unused

(4) Bits	Switches
0-6	Unused
7	FRGNSW

P1TEXT  
(TIB 05)

Purpose

Save input P1-text elements and newly created Debug-text elements.

1	Variable
Number of bytes following	P1A-text element

Phases Involved

Phase 35 builds this table and makes entries in this table as it is processing its input once the determination has been made that no more debug declaratives exist. Phase 35 uses this table to accumulate verb strings until all debug processing is complete. Phase 35 deletes this table when processing is complete.

QALTB (Report Writer)  
(TIB 23)

Purpose

Work table, for example, to store qualified names and qualifiers in reverse order of appearance in RD.

Variable
Name(s) in EBCDIC, each preceded by 1-byte count

Entry Frequency

One for each name in the current string of a name and its qualifiers.

Phases Involved

Phase 12 builds this table as it scans a name and its qualifiers.

Phase 12 uses this table to reverse the order of a name and its qualifiers.

QFILE  
(TIB 23)

Purpose

Store Q-Routine GN numbers connected with CD entries or with QSAM, QISAM, or BISAM files in which at least one of the records contains an OCCURS DEPENDING ON clause.

2	2
DCB number for first file or CD number	GN number for single Q-Routine for file or for all Q-Routines for CD entry, or for chain of GNS, where there is more than one Q-Routine associated with a file.
①	②

Entry Frequency

One entry for each CD entry file or of this type.

Phases Involved

Phase 22 builds this table from Q-Routines for files or CD entries.

Phase 21 increments the DCB number for DCBs for checkpoint files.

Phase 3 adds Q-Routine GN numbers to dictionary attributes for corresponding files or CD entries.

- 1) for SD  
 byte 1 = 09  
 byte 2 = BLL#
- 2) Table ends with a word of zeros.

QGNTBL  
(TIB 24)

Purpose

Pass Q-Routine GN numbers and their Procedure block numbers from phase 63 to phase 64. This table is built only if there is a Q-BEGIN macro element passed in Procedure A-text.

2	1
GN number	Procedure block number

Entry Frequency

One for each Q-Routine GN.

Phases Involved

Phase 63 builds this table from the GNLBTBL.

Phase 64 uses the data to initialize Q-Routines during processing for INIT3.

QITBL  
(TIB 22)

Purpose

Contains a pointer to the dictionary attributes of each OCCURS DEPENDING ON object entered in the OD2TBL table.

4	4
Pointer to entry in dictionary for OCCURS DEPENDING ON variable	Displacement of entry in OD2TBL for OCCURS DEPENDING ON variable

①

Entry Frequency

One entry for each OD2TBL entry.

Phases Involved

Phase 22 builds this table from dictionary and OD2TBL.

Phase 22 combines this table with OD2TBL and QRTN tables to form QVAR table.

Phase 25 uses this table with QRTN and OCCTBL tables to build the ODOTBL table.

- ① Table ends with a word of zeros.

QLTABL

**Purpose**  
Store qualified names in reverse order of receipt (which was previously reversed).

1	1	Variable	1	Variable
35	00	Qualified name in EBCDIC	Number of bytes in preceding field	First qualified name in EBCDIC

**Entry Frequency**  
One-entry table containing name currently being processed and its qualifiers.

1	Variable	1
Number of bytes in preceding field	Last qualifier name in EBCDIC	Number of bytes in preceding field

**Phases Involved**  
Phase 10 builds this table when qualified EBCDIC name is inserted in QNMTBL or OD2TBL.  
Phases 12 and 1B build this table when qualified EBCDIC name is written on PO-text.

QNMTBL (TIB 2)

**Purpose**  
Store EBCDIC names <sup>①</sup> from Data and Environment divisions until Data IC-text is written.

1	1	Variable	1	Variable
n	c	Last qualifier name in EBCDIC	c	Second from last qualifier-name in EBCDIC

**Entry Frequency**  
One entry for each name of this type.

Variable	1
Qualified name in EBCDIC	00

**Phases Involved**  
Phase 10 builds this table from clauses in FD, SELECT, and APPLY statements.  
Phase 10 uses this table to write out Data IC-text.

① For REPORT, LABEL RECORD, ACTUAL KEY, NOMINAL KEY, RECORD KEY, TOTALING-AREA, TRACK AREA, and APPLY REORG-CRITERIA clauses.

QRTN (TIB 21)

**Purpose**  
Store GN numbers of Q-Routines and OD2TBL pointers for each OCCURS DEPENDING ON clause in a record.

2	2	4
GN number of first Q-Routine	Number of fields that follow	Displacement of OD2TBL entry for the first Q-Routine

**Entry Frequency**  
One entry for each record containing an OCCURS DEPENDING ON clause.

4
Displacement of OD2TBL entry for last Q-Routine

**Phases Involved**  
Phase 22 builds this table from GNCTR in COMMON and OD2TBL.

Phase 22 combines this table with OD2TBL and QITBL tables to form the QVAR table.  
Phase 25 uses this table with QITBL and OCCTBL tables to build the ODOTBL table.

SBL  
(TIB 25)

Purpose  
Hold and transmit data on secondary base locators (SBLs).

1
First SBL number associated with an INCRA verb

Entry Frequency  
One entry for each INCRA verb generated.

Phases Involved  
Phase 22 creates and passes this table to phase 3.  
Phase 3 uses this table to generate for each INCRA verb a P1-text literal the value of which equals the number of SBLs associated with the verb.

TBL  
(TIB 3)

Purpose  
Store GN numbers for Q-Routines during Data A-text processing.

2
GN number of Q-Routine

Entry Frequency  
One entry for each Q-Routine definition in Data A-text.

Phases Involved  
Phase 6 or, under the optimizer version of the compiler, phase 64 makes an entry in this table when it finds a Q-Routine identification element in Data A-text.  
Phase 6 or phase 64 uses this table to initialize Q-Routines when generating INIT3 code for object module.

QVAR  
(TIB 24)

Purpose  
Store Q-Routine GN numbers connected with items containing OCCURS DEPENDING ON clauses.

4	2	4
Pointer to dictionary entry for object of OCCURS DEPENDING ON for first item	GN number for Q-Routine for first item	Dictionary pointer for last item

Entry Frequency  
One entry for each item.

Phases Involved

Phase 2 builds this table from QRTN, QITBL, and OD2TBL tables.

2	4
GN number for last item	
	0

Phase 3 adds Q-Routine GN numbers to dictionary attributes of items that are objects of OCCURS DEPENDING ON clauses.

①	<u>Bits</u>	<u>Contents</u>
	0-9	Zeros
	10-22	Dictionary section number
	23-31	Displacement in section

RCDTBL  
(TIB 11)

Purpose

Store each 01-level record-name in FD Section until input/output verb processing.

2	1	Variable
Displacement of entry for file in FNTBL		Record-name in EBCDIC

Entry Frequency

One entry for each 01-level entry in FD section.

Phases Involved

Phase 10 builds this table from 01-level statements. Phase 1B uses this table in processing input/output statements.

RDFSTK  
(TIB 11)

Purpose

Store information about subject and object of REDEFINES clauses.

1	3	4	4
Level number of REDEFINES subject	Contents of address parameter field (idk)	Unused	Length of REDEFINES object

Entry Frequency

One entry for each REDEFINES clause.

Phases Involved

Phase 22 builds this table from dictionary. Phase 22 uses this table to assign address parameter to items after REDEFINES clause.

① First entry is a dummy

RENAMTB  
(TIB 3)

Purpose

Store information for associating a renamed item with all of its renaming items if the SYNDMP or TEST option is specified.

3	3
Dictionary pointer for renamed item	Dictionary pointer for renaming item



Entry Frequency

On entry for each RENAMES item.

Phases Involved

Phase 22 builds this table while processing RENAMES clauses.

Phase 25 uses this table to associate renamed items with renaming items.

REPTAB  
(TIB 29)

Purpose

Store COPY...REPLACING data-names to be used during read from library.

1	Variable	1	Variable
c	Word being replaced	c	Replacing word, literal, or identifier.

①

Entry Frequency

One entry for each pair specified in the REPLACING clause of the COPY statement currently being executed.

Phases Involved

Phase 02 builds this table from COPY...REPLACING clause in source program.

Phase 02 uses this table to replace data-names while source statements are being read from a library.

RLDTBL (Phase 6)

②

Purpose

Store information on items to be inserted in data area or a Global Table in the object module.

1	3	1	3
84 = DCB address or other Data A-text element	Target address	Priority (0, if no segmentation)	Value of adcon
94 = GN or PN definition			
A4 = VN definition			
C0 = TGT adcon			

Entry Frequency

One entry for each PN definition, VN definition, and adcon in Data A-text.

Phases Involved

Phase 6 builds this table during Data A-text processing.

Phase 6 uses this table to write RLD and text cards for object module.

RLDTBL (Phases 63 and 64)

(TIB 28)

Purpose

Store information on items to be inserted in the data area or a Global Table in the object module.

1	3	1	3
Code	Target	Priority	Value of
defining	address	(0, if no	ADCON
item ③		segmentation)	

Entry Frequency

One entry for each PN and GN definition, VN definition, and each address constant in Data A-text.

Phases Involved

Phases 63 and 64 build this table during Procedure A-text and Data A-text processing.

Phase 64 uses this table to write RLD and text cards for the object module and to resolve VN EQUATE PN and VN EQUATE GN addresses.

① Last entry in the table is followed by a byte of zeros.

② There is no TIB for this table. Phase 6 uses the phase 00 routine GETALL to get space, but moves data in and out of the table by itself. When it receives control back from phase 00, phase 6 stores the first address in the table space in location ARLDTB and the length in bytes in location RLDSIZ. During processing, it uses location RLDINDEX as a counter of the bytes used so far.

③ Code	Meaning
10	RPT-ORIGIN GN
84	DCB address or other Data A-text element
94	GN or PN definition
A4	VN definition
C4	INIT1 ADCON address in TGT

RNMTBL (Phase 22)

(TIB 12)

Purpose

Store information on objects of REDEFINES clauses.

Entry Frequency

One entry for each REDEFINES clause.

1	3	1
Level number	Pointer to	Dictionary
of REDEFINES	dictionary	minor code
subject	entry for	for object
	REDEFINES	
	object	

Phases Involved

Phase 22 builds this table from dictionary.

Phase 22 uses this table to check whether the REDEFINES clause is valid.

RNMTBL (Report Writer)

(TIB 12) Purpose  
Store data-names of report groups.

Entry Frequency  
One for each REPORT GROUP that has a data-name and is not a detail report group.

Phases Involved  
Phase 12 builds this table from scan of report groups.  
Phase 1B uses this table to generate coding in response to USE BEFORE REPORTING statements.

32	2	1
Data-name for report group in EBCDIC (1)	GN number for this report-group	<u>NOP code</u> 00 = NOP PLUS 01 = NOP ZERO (2)

- (1) Left-justified, padded with binary zeros in low-order bytes.
- (2) First entry is a dummy.

ROLTBL (Report Writer)

(TIB 15) Purpose  
Store the SUM clause data-names and operand-names that are needed to create ROL-ROUT and RST-ROUT.

Entry Frequency  
One entry for each sum rolled forward.

Phases Involved  
Phase 12 DOROL routine builds this table.  
Phase 12 GNSPRT routine uses this table to create ROL-ROUT and RST-ROUT.

2	1	2	2	4
n (1)	SUM level	Unused	Displacement of SUM name entry in SNMTBL (this is the item to be rolled forward)	Displacement of SUM name SNMTBL (2) or nnnn portion of S.-name (this is the item into which the sum is to be rolled)

4	4
Displacement of SUM name in SNMTBL (2) or nnnn portion of S.name (this is the item into which the sum is to be rolled)	Contains zeros to indicate end of entry

- (1) Contains zeros if this is the last entry in the table
- (2) 

Bytes	Contents
0	FF
1	00
2-3	Displacement

ROUTBL (Report Writer)  
(TIB 16)

Purpose  
Store GNs for routines in each Report Writer generated program.

Entry Frequency  
One entry for each report in Report Section.

Phases Involved  
Phase 12 builds and uses this table.  
Phase 1B uses this table in Report Writer verb processing.

2	2	2	2	2
GN number for RPH-ROUT	GN number for RPF-ROUT	GN number for PGH-ROUT	GN number for PGF-ROUT	GN number for 1ST-ROUT

2	2	2	2	2
GN number for LST-ROUT	GN number for WRT-ROUT	GN number for WRT-1	GN number for WRT-2	GN number for CTB-ROUT

2	2	2	2	2
GN number for ROL-ROUT	GN number for RST-ROUT	GN number for RST-1	Unused	GN number for detail now being processed it is moved to DETBL at end of processing for the detail

2	2	2	2	2
GN number for CHF-ROUT	GN number for CFF-ROUT	GN number for PH-1	GN number for ROL-1	GN number for INT-ROUT

2	2	2	2
GN number for ALS-ROUT	GN number for RLS-ROUT	GN number for SAV-ROUT	GN number for RET-ROUT

RUNTBL  
(TIB 35)

Purpose  
Collect and condense information about checkpoints.

Entry Frequency  
One entry for each valid RERUN statement specified with the "integer-1 RECORDS" option.

2	2	4
DCB number of RERUN file	DCB number of checkpoint file	"Integer" for RERUN file

Phases Involved

Phase 21 builds this table from the CKPTBL table.

Phase 51 uses this table to generate coding to count and test "integer", for READS and WRITES for RERUN files.

RWRTBL (Report Writer)  
(TIB 13) Purpose

Store information about a report-name.

Entry Frequency

One entry for each report-name.

30	5	2	2
Report-name in EBCDIC	-mnn portion of record-name for file-name-1	Pointer to file-name-1 entry in FNTBL	Size in binary of larger record
①			

Phases Involved

Phases 10 and 12 build this table from scan of FD and RD entries.

Phase 1B uses this table during scan of Report Writer verbs.

5	2	2
-nnnn portion of record-name for file-name-2	Pointer to entry for file-name-2 in FNTBL	Displacement in ROUTBL of entry for this report
②	②	

- ① Left-justified, padded with binary zeros in low-order bytes.
- ② Contains zeros, if report is to be written on only one file.

SAMETB

(TIB 19) Purpose

Save information on SAME AREA files until buffer allocation and Data A-text generation of DCB and DECB elements.

Entry Frequency

One entry for each file named in a SAME AREA clause.

1	3	2	2
SAME AREA number	Relative location in object module of DCB, DECB, or FIB	Number of buffers for queued file	Block size rounded up to multiple of eight
	①	②	②a

Phases Involved

Phase 21 builds this table from Data IC-text.

Phase 21 uses this table to allocate buffers and fill in fields for DCBs and DECBs.

1	1	4 bits	4 bits	1
BL number for this file	Key length		Access method	
	②b	③	④	⑤

SATBL  
(TIB 5)

Purpose  
Store file-names associated with SAME AREA clauses until all SAME clauses have been processed.

1	3	1	30	31
Count of files in entry	Card number of entry	c	File-name in EBCDIC	Repeat file-name and count fields once for each file name in clause

Entry Frequency  
One entry for each file in a SAME AREA clause.

Phases Involved  
Phase 10 builds this table from SAME AREA clauses in source program.  
Phase 10 uses this table to check SAME AREA clause syntax.

- ① Contains the relative location of the DECB for BASIC files except for BASIC files containing spanned records. For BASIC files containing spanned records and all other files, this field contains the relative location of the DCB. For VSAM files, the field contains the relative location of the FIB.
- ② Field = 1 if this is a BASIC file.  
= 1 if this is a VSAM file.
- ②a Maximum record size rounded to a multiple of 8 for VSAM file.
- ②b Field is set to zero for VSAM file.

④	<u>Code</u>	<u>Access Method</u>
	0000	QSAM
	0001	QISAM
	0010	BISAM
	0100	BSAM
⑤	<u>Bit</u>	<u>Meaning, if on</u>
	0	QSAM file containing spanned records
	1	BASIC file containing spanned records
	2	VSAM file
	3-7	Unused

③	<u>Bit</u>	<u>Meaning</u>
	0	Format V
	1	Spanned records, ADVANCING or SAME RECORD AREA specified
	2	Direct BDAM file when REWRITE is used
	3	Direct BDAM or direct BSAM file with key

SEGINDX  
(TIB 16)

Purpose  
Store information about program fragments if the SYNDMP or STATE option is specified

1	3	3	3
Priority number	Address of this fragment relative to the beginning of the segment	Table-locator for PROCTAB entry for first card/verb in this fragment	Table-locator for PROCTAB entry for last card/verb in this fragment
		①	①

Entry Frequency  
One entry for each program fragment.

Phases Involved  
Phase 65 builds this table while reading Debug-text and building the PROCTAB table. COBOL library subroutines use this table.

- ① For the SYNDMP option, the field contents are:

<u>Bits</u>	<u>Contents</u>
0-14	Relative block number in PROCTAB
15-23	Displacement within block

For the STATE option, the field contents are:

<u>Bits</u>	<u>Contents</u>
0-23	Displacement from the beginning of the PROCTAB entries in the object module

SEGTBL  
(TIB 15)

Purpose  
Store disk addresses of sections of Procedure A-text.

1	4
Priority number	Device address ①

Entry Frequency  
One entry for each segment control break encountered in P2-text.

Phases Involved  
Phase 51 creates an entry when it finds a segment control break. Gets priority from PNOUT + 1 in phase 5, and device address from cell SEGSAV in phase 00. Phase 6 or, under the optimizer version of the compiler, phases 62 and 63 use this table to combine sections into a segment.

① <u>Byte</u>	<u>Contents</u>
0-1	Relative track number
2	Block number
3	Zeros

SETTBL  
(TIB 21)

Purpose  
Store operands before TO, UP, or DOWN in SET statement; store receiving fields before ON SIZE ERROR or next verb in ADD or SUBTRACT statements with multiple receiving fields.

Variable
Element as encountered in P1-text

Entry Frequency  
One entry for each element of P1-text encountered before desired element is found in input buffer.

Phases Involved  
Phase 4 builds this table while processing SET statements, or ADD or SUBTRACT statements with more than one storage field. Phase 4 uses this table to temporarily bypass operands in SETTBL table while it scans input buffer for desired element following these operands.

SMRCDTBL  
(TIB 5)

Purpose  
Save information on SAME RECORD AREA files until buffer allocation and Data A-text generation.

1	3	2	1	1
Same RECORD AREA number	Relative location of DCB or DECB for BASIC FILES or FIB	Maximum record size	First BL number for file	Flag byte

Entry Frequency  
One entry for each file named in a SAME RECORD AREA clause.

Phases Involved

Phase 21 builds this table from Data IC-text and PIOTBL.

Phase 21 uses this table to allocate record areas, to generate block address definitions, and to fill in DCB or DECB fields.

1	1
Flag byte	Key length (if bit 3 is on in preceding field)

SMSTBL (Report Writer)  
(TIB 28)

Purpose  
Store SUM clause operand-names for correlation of SUM and SOURCE clauses.

2	c
c	SUM clause operand-name in EBCDIC

Entry Frequency  
One for each operand-name in a SUM clause.

Phases Involved

Phase 12 builds this table from SUM clauses. Phase 12 uses this table with SRCTBL and SUMTBL to generate USM-ROUT routine for each detail report group and to build ROLTBL.

① Contains the relative location of the DCB for BSAM files containing spanned records. For all other BASIC files, this field contains the relative location of the DECB. For VSAM files, this field contains the relative location of FIB.

Bits	Meaning, if on
0	Format V
1	QSAM with spanned records and/or ADVANCING option
2	Unused
3	File is direct BSAM or direct BDAM
4-7	Access method
	0000 = QSAM
	0001 = QISAM
	0010 = BISAM
	0100 = BSAM
	1000 = BDAM

Bit	Meaning
0	QSAM file containing spanned records
1	BSAM file containing spanned records
2	VSAM file
3-7	Unused

④ First entry is a dummy.



SNMTBL (Report Writer)  
(TIB 35)

Purpose  
Store all data-names of SUM clauses.

32	3
Data-name for sum bucket ①	Unused ②

Entry Frequency  
One entry for each SUM clause.

Phases Involved  
Phase 12 builds this table from SUM clause.  
Phase 12 uses this table to correlate SOURCE and SUM clauses, build ROITBL, and generate USM-ROUT routine.

SPNTBL  
(TIB 21)

Purpose  
Store function-name information from SPECIAL-NAMES paragraph in Environment Division.

3	1	Variable ③
3	c	Mnemonic-name in EBCDIC

Entry Frequency  
One entry for each function-name.

Phases Involved  
Phase 10 builds this table from SPECIAL-NAMES paragraph.  
Phases 12 and 1B use this table to substitute function-name for mnemonic-name in Procedure Division.

- ① Left-justified, padded with binary zeros in low-order bytes.
- ② First entry in the table is a dummy.
- ③ Three possible configurations:

	<u>Byte 0</u>	<u>Byte 1</u>	<u>Byte 2</u>
a.	1-character literal in EBCDIC	Unused	Unused
b.	54	Code for COBOL word used (see note 7 under P0-text in Appendix C).	Unused
c.	55	Code for carriage control word (see note 5 under P0-text in Appendix C).	Unused

SRATBL  
(TIB 6)

Purpose  
Store file-names associated with SAME RECORD AREA clauses until all SAME clauses have been processed.

1	3	1	30	31
Count of files in entry	Card number of entry	c	File-name in EBCDIC	(Repeat file-name and count fields once for each file in clause)

Entry Frequency  
One entry for each file named in a SAME RECORD AREA clause.

Phases Involved  
Phase 10 builds this table from SAME RECORD AREA clauses in source program. Phase 10 uses this table to check SAME RECORD AREA clause syntax.

SRCHKY  
(TIB 34)

Purpose  
Save names of keys cited in KEY clause for group item until group item is processed.

2	1	Variable
01 = ASCENDING 02 = DESCENDING	c	Name of key in EBCDIC

Entry Frequency  
One entry for each key named in KEY clause in current group item.

Phases Involved  
Phase 22 builds this table from group items in Data IC-text. Phase 2 uses this table to make sure keys named are defined in group. If not, sets error bit in INDKEY table for phase 3 reference.

SRCTBL (Report Writer)  
(TIB 22)

Purpose  
Store SOURCE clause operand names to correlate SOURCE and SUM clauses.

2	2	Variable
Length of SOURCE clause operand-name	Displacement into DETBL of detail report group data-name	SOURCE operand with all qualifiers, indexes, and subscripts (if any) in P0-text format.

Entry Frequency  
One entry for each SOURCE clause in each detail report group.

Phases Involved  
Phase 12 builds this table while scanning detail report groups. Phase 12 uses this table in conjunction with SMSTBL and SUMTBL to generate SUM-ROUT routine for each detail report group.

[N  
3 5)

Purpose  
Save all subscript strings preceding an UNSTRING verb string.

Entry Frequency  
One entry for each subscripted data-name in an UNSTRING statement

Phases Involved  
Phase 45 builds and uses this table.

3	Variable	Variable
8439 (hex) followed   by count of elements   that follow	Data-name   reference (30)   element	Data-name reference (30)   or alphanumeric literal   (34) element for first   subscript string

Variable	Variable	Variable
Data-name reference   (30) or alphanumeric   literal (34) element   for second subscript   string	Data-name reference   (30) or alphanumeric   literal (34) element   for third subscript   string	Subscripted data-   name reference (31)   element with sub-   script string ID   number in place of   idk field

OUT  
B 11)

Purpose  
Store subscript strings in the order in which they are to be written as P2-text. ①

Entry Frequency  
The table is divided into sections which contain the subscript strings for any subscripted data items that are referred to by the text in the corresponding section of the TXTOUT table.

Phases Involved  
Phase 45 builds and uses this table with the TXTOUT table.

3	Variable	Variable
8439 (hex) followed   by count of elements   that follow	Data-name   reference (30)   element	Data-name reference (30)   or alphanumeric literal   (34) element for first   subscript string

Variable	Variable	Variable
Data-name reference   (30) or alphanumeric   literal (34) element   for second subscript   string	Data-name reference   (30) or alphanumeric   literal (34) element   for third subscript   string	Subscripted data-   name reference (31)   element with sub-   script string ID   number in place of   idk field

2
FFFF (hex)   to indicate   end of subscript   information

If the corresponding data item in the TXTOUT table is not subscripted, the entry consists of a halfword containing hex 'FFFF'.

SSDELIM  
(TIB 20)

Purpose  
Save delimiters from an UNSTRING statement that have to be repeated.

Entry Frequency  
One entry for each delimiter that is either subscripted or follows a variable-length group.

Phases Involved  
Phase 45 builds and uses this table.

2	2	2	Variable	14
Pointer to entry in SSCIN for subscript string (if none contains zeros)	541D (hex) - code for "DELIMITED BY"	5479 (hex) - code for "ALL" if specified	Data-name reference (30) element	Data-name information for UNSTRJ (2A) element

STRING  
(TIB 9)

Purpose  
Store verb strings while they are being built for output as P2-text.

Entry Frequency  
One entry for each operand in current string.

Phases Involved  
Phase 4 builds this table as strings are processed. Phase 4 uses this table to collect output before generating.

Variable	Variable	Variable
Verb	① First operand in string	① Last operand in string

SUMTBL (Report Writer)  
(TIB 19)

Purpose  
Store data-names and operand-names from SUM clauses that are used to create USM-ROUT, INT-ROUT, RST-ROUT, and ROL-ROUT routines.

Entry Frequency  
One for each SUM clause.

Phases Involved  
Phase 12 builds this table from scan of SUM clauses. Phase 12 uses this table to build USM-ROUT, INT-ROUT, RST-ROUT, and ROL-ROUT routines.

2	1	3	1
n	SUM level	Card number for this SUM clause	Reset level
②			

2	1	4
Displacement of entry in DETTBL for detail name in SUM...UPON clause	Code for next field	Pointer to SUM name in SNMTBL or nnnn portion of S.-name
	③	

7	24	2
E.-name (REDEFINES) in P0-text format	PICTURE for name in EBCDIC	Displacement of entry in SMSTBL for first operand-name in SUM clause
④		

	2	2
	Displacement of entry in SMSTBL for last operand-name in SUM clause	Zeros

Number of bytes in preceding field.

Contains zeros if this is last entry in table.

Code	Meaning	Bits	Contents
00	Next two bytes contain displacement into SNMTBL.	0	06
10	Next four bytes contain nnnn portion of S.-name.	1	E
FF	Next field contains nnnn portion of S.-name.	2	. (period)
		3-6	nnnn

TABLE

Purpose

Store information needed by phase 65 for processing the SYMDMP, TEST, STATE, and FLOW options. Also used to pass information between phases 62, 63, and 64.

Entry Frequency

Information entered depends upon options in effect. Only the first five fields are filled in if phase 60 builds the table.

Phases Involved

Phases 60 or 62, 63, and 64 build this table. Phase 65 uses this table in processing the SYMDMP, STATE, and FLOW options.

	2	4	4	4	4	4
Displacement of DEBUG TABLE in TGT	Relative address of byte after INIT3	Relative address of first instruction in Procedure Division (in root segment when program is segmented)	Relative address of START of Q-Routines or INIT2 if no Q-Routines	Address of START of Q-Routines	Note address of first block of code generated for independent segments on SYSUT1	
(1)	(3) (4)	(3) (4)	(3) (4)	(3) (4)	(3) (4)	(3) (5)

	2	2	2
Displacement of ILBOFLW0 virtual from beginning of PGT if FLOW and DYNAM/RESIDENT used by phase 64	Displacement of ILBOTEF3 virtual from beginning of PGT if SYMDMP and DYNAM/RESIDENT and program contains floating-point item used only by phase 64	Displacement of ILBOSGM0 virtual from beginning of PGT if TEST option is specified for a segmented program	
(1)	(1)	(2)	

2
Length of transient area if program is segmented-used by phase 64 to fill in TALENGTH field in TGT

- ① Filled in by phase 62.
- ② Filled in by phase 63.
- ③ Filled in by phase 64.
- ④ Filled in only if the SYMDMP or the STATE option is in effect.
- ⑤ Contains zeros if the program is not segmented.

TIBP (Alternate name for RNMTBL)

TIBR (Alternate name for RDFSTK)

TOTTBL  
(TIB 32)

Purpose  
Store TOTALED option information from Data Division for use in Procedure Division processing and by phase 22.

1	2	1	30
02	Displacement of entry for associated file in FNTBL table	c	Data-name-3 from TOTALED option in EBCDIC form
			①

Entry Frequency  
One entry for each TOTALED option specified.

Phases Involved  
Phase 10 builds this table from source program FD entries.  
Phase 22 uses this table to assign BLLs to a TOTALED AREA data-name and items subordinate to it.

- ① Padding starts from low-order bytes.

TXTOU  
(TIB 19)

Purpose  
Save UNSTRING verb strings.

Entry Frequency  
One entry for each UNSTRING verb string. The table is organized in sections which contain all the strings, except subscript strings, for an UNSTRING statement.

3	2	3	Variable
8465 (hex) followed by count of elements that follow	5496 (hex) - code for "FIRST"	Verb information (24) element containing the sequence number of this string and the total number of strings	Text string element for DELIMITER, RECEIVING FIELD, DELIMITER IN, COUN IN, POINTER, or TALLYING

2	2	2	3	3
54A1 (hex) - code for "END"	FFFF (hex) - to indicate end of string	8484 (hex) - ①	Numeric literal (BB) element which contains the number of Q-routines needed ①	GN reference (AA) element which contains the number of the first Q-routine ①

2
FFFF (hex) - to indicate ① end of string

- ① The field is present only if the text string is for a data item which is the object of an OCCURS...DEPENDING ON clause.

**UPSTBL**  
(TIB 29)

Purpose  
Build during scan by phase 10 from SPECIAL-NAMES (if not used, must be released).

1	3	1	6	Variable
n	card number of UPSI-X	06	UPI-X name	C Mnemonic name
①	②			③

Phases Involved  
Phase 22 deletes this table after use.

- ① Count of bytes in UPSTBL entry.  
② Card number of UPSI definition  
③ Type of entry and count.

1	Variable	1	Variable
C	condition-name	C	condition-name
③		③	

Bit      Meaning  
0-1      Type of entry  
         11 = Mnemonic name  
         10 = ON condition name  
         01 = OFF condition name  
2-7      Count of name which follows

**USETBL**  
(TIB 26)

Purpose  
Associate USE DECLARATIVE information with procedure name.

2	1
PN number	Attributes
	②

Entry Frequency  
One entry for each USE sentence specified in DECLARATIVES Section.

Phases Involved  
Phase 1B builds this table from source program USE DECLARATIVE entries.  
Phase 51 uses this table to generate variable entry code in DECLARATIVES.

- ② Bits      Meaning, if on  
0      Unused  
1      MULTIPLE FILE  
2      AFTER  
3      BEFORE  
4      REEL/UNIT  
5      FILE  
6      BEGINNING  
7      ENDING

USNGTBL  
(TIB 2)

Purpose

Store dictionary pointer and PNs for Error or Label Declarative associated with the USING clause of SORT or MERGE verb until all file-names in clause have been processed.

0-3	4-13	14-15
Dictionary pointer	PNs for Error or Label Declarative	Unused

Entry Frequency

One entry for each file-name in SORT...USING clause. One entry for each file-name in MERGE...USING clause.

Phase Involved

Phase 30 builds and uses this table during USING processing.

VALGRP  
(TIB 6)

Purpose

Save Data A-text address constant definitions for group items containing VALUE clauses.

1	1	1	3	1	2
10	c	28	Relative address in object module where address constant is located	Size of	item
2				3	

Entry Frequency

One entry for each group item with a VALUE clause that is current by being processed.

1	Variable
c	Alphanumeric constant itself
1 4	

Phases Involved

Phases 20 builds this table from Data IC-text LD entries.

Phase 22 adds the length of the group item and, if it is not an ALL constant, deletes the literal byte count.

VALTRU  
(TIB33)

Purpose

Store literals for VALUE...THRU clause in level-88 group item.

1	Variable	1	Variable	1
c	P1-text element for literal	5	c	P1-text element for literal
5	6	5	6	7

Entry Frequency

One entry for each VALUE clause of this type.

Phases Involved

Phase 20 builds this table from Data IC-text LD entries.

Phase 22 uses for syntax checking of the VALUE IS SERIES clause.

Phase 3 uses this table to fill in P1-text literals with the actual values.



- ① Data A-text prefix
- ② Number of bytes to follow.
- ③ Type of constant as follows:  
 01 = Alphanumeric  
 FF = ALL constant
- ④ If not ALL constant this byte is deleted by phase 22.
- ⑤ In the high-order four bits:  
 0 = Value is not followed by THRU  
 8 = Value is followed by THRU  
  
 The low-order bits contain the count of bytes in the next two fields.
- ⑥ First byte contains the following code:  

Code	Type of Literal
32	Numeric
33	Floating-point
34	Alphanumeric
39	ALL constant
75	ALL constant (one byte only)
- ⑦ Signifies the end of a series.

**VARLTBL**  
(TIB 15)

Purpose  
 Store information about variable-length items needed for the DATATAB table if the SYMDMP or TEST option is specified.

3	3
Dictionary pointer for	Maximum size (including variable-length items
	slack bytes) in bytes

Entry Frequency  
 One entry for each variable-length item.

Phases Involved  
Phase 22 builds this table using information in the GPLSTK table.  
Phase 25 uses this table while processing variable-length items for the DATATAB table.

**VARYTB**  
(TIB 1)

Purpose  
 Control GN numbers branched to in PERFORM...VARYING statement.

3	3	3	3
GN number for	VN number	GN number for	GN number
condition	for varied	ADD verb string	for MOVE GN
branch	branches	that increments	

Entry Frequency  
 One entry for each PERFORM...VARYING statement.

Phases Involved  
Phase 4 builds this table from PERFORM...VARYING strings in P1-text.  
Phase 4 uses this table to issue P2-text strings with correct branches for different steps.

VIRPTR  
(TIB 13)

Purpose  
Store pointers to CVIRTB during virtual optimization.

2
Displacement from start of PGT in the object module to virtual

Entry Frequency  
One entry for each virtual definition element.

Phases Involved  
Phase 6 or, under the optimizer version of the compiler, phase 62 builds this table when it builds CVIRTB. Phase 6 or phase 62 uses this table with CVIRTB to eliminate duplicate virtuals. After PGT allocation, each entry points to entry in PGT virtual field. Phase 64 uses this table to locate the constant (EBCDIC name) to be inserted in the object program listing, if the listing is requested.

VNPNTBL  
(TIB 29)

Purpose  
Establish addressability at PN definition location.

1	2	2
Type ①	PN or GN number	VN number

Entry Frequency  
One entry for each VN EQUATE PN or VN EQUATE GN element encountered during Optimization A-text processing.

Phases Involved  
Phase 62 builds this table during Optimization A-text processing. Phase 62 uses this table to update the ACCUMCTR counter by 4 for each load instruction to be generated by phase 63 for each PN or GN associated with an ALTER statement. Phase 63 creates, for every entry in this table, RLD entries for the VNI cells in the PGT. The phase generates a load instruction of the current Procedure Block into register 11 at the point of definition of the PN or GN associated with an ALTER statement.

①	<u>Code</u>	<u>Meaning</u>
	00	PN, ALTER
	0F	PN, PERFORM
	F0	GN, ALTER
	FF	GN, PERFORM

(code values are in hexadecimal)

VNPTY  
(TIB 17)

Purpose  
Store VN numbers and associated priority numbers to later compute the position of VNI cells in the object module.

1	2
Priority number	VN number

Entry Frequency  
One entry for each VN number.

Phases Involved  
Phase 6 or, under the optimizer version of the compiler phase 62 builds this table from segmentation elements in Optimization A-text.  
Phase 6 or, under the optimizer version of the compiler, phase 64 sorts entries by priority numbers and uses the resulting order to compute the position of VNI cells in the object module.

VNTBL  
(TIB 11)

Purpose  
Store information on procedure-names that have been altered by an ALTER statement or are ends-of-range of PERFORM statements.

2	2
PN number	VN number corresponding to PN

Entry Frequency  
One entry for each procedure-name.

Phases Involved  
Phase 4 builds this table from P1-text PNs and VNCTR in COMMON.  
Phase 4 uses this table to modify addresses and set up return VNs.

VRBDN  
(TIB 07)

Purpose  
Describes each data item encountered by the phase in the P1-text for a specific verb for debugging.

1	3	2
Switch	Dictionary pointer	Displacement in DBGTXT
①		

Phases Involved  
Phase 35 builds this table and uses it to describe each data item encountered by the phase in the P1-text for a specific verb which may be considered for debugging. Phase 35 deletes this table upon completion of processing.

2
Displacement in P1TEXT

Bit	Meaning
0	Valid entry
1	Data item in DTAB table
2	Generate data item debug text if statistics is on and duplicates is off
3	Generate debug text twice
4	Data item may change
5	Duplicate Data item
6	Data item in DTAB, subscripted

VRDEFTBL  
(TIB 14)

Purpose

Store information about the occurrences of COBOL verbs.

1	1	2	1	1	1
48	00	Number of occurrences	E0	Alphabetic verb sequence number	C0

Entry Frequency

One entry for each COBOL verb used.

1	1	Variable	1
Length	FB	Verb-text	FF

Phases Involved

Phase 1B builds this table when VBREF or VBSUM is specified.

Phase 22 uses this table to generate verb DEF-text.

XAVAL  
(TIB 2)

Purpose

Optimize use of arithmetic temporary storage by object module.

2
ID number of 8-byte slot available in Working-Storage

Entry Frequency

One entry for each 8-byte slot.

Phases Involved

Phase 50 creates an entry for each slot as it is released.

Phase 50 uses this table to obtain temporary storage that has been used and released in the object module.

XINTR  
(TIB 1)

Purpose

Store and analyze intermediate results in compile-time arithmetic.

16	2	2
Compile-time value in internal decimal	Length after scaling in internal decimal	Length after scaling in binary

Entry Frequency

One entry for each intermediate result.

Phases Involved

Phase 50 builds this table from ID number of intermediate result passed from phase 4 and its own analysis of operands in arithmetic statements.

2	2	2	2
Number of digits after scaling	Number of decimal places after scaling	Length occupied in Working-Storage	Relative pointer in Working-Storage

Phase 50 uses this table to process compile-time arithmetic verbs.

1	1	2
Register number	Characteristics of operand ①	Intermediate result number

- Bit Meaning, if on**
- 0 Register used in double-precision mode
  - 1 Overflow could occur
  - 2 Double-precision floating-point
  - 3 Operand is in register
  - 4 Operand is a literal
  - 5 Operand is floating-point
  - 6 Operand is generated constant
  - 7 Operand is literal ZERO

CRPT  
[B 3)

**Purpose**  
Store subscript and index information for optimization.

**Entry Frequency**  
One entry for each subscripted or indexed item.

**Phases Involved**  
Phase 50 builds this table from subscript verb strings passed by phase 4.  
Phase 50 uses this table to calculate address of subscripted or indexed item, or to generate object code for the calculation.

2	2	1	3	4
n+1	Number of subscripts or indexes	0	New addressing parameter of subscripted or indexed item ①	Dictionary pointer to unique identifier of subscripted or indexed item ②

1	3	3 (optional)	3 (optional)
Flag byte ③	Dictionary pointer to unique identifier of first subscript or index-name ④	Flag byte ③	Dictionary pointer to unique identifier of second subscript or index-name ④

3 (optional)	3 (optional)
Flag byte ③	Dictionary pointer to unique identifier of third subscript or index-name ④

- ① Bits    Meaning  
 0-3       3 = Byte 2 contains number of register which contains new address at object time.  
           6 = Bytes 2 and 3 contain the number of a SUBSCRIPT CELL which contains the new address at object time.

- ② Bits        Contents  
       0-9        Zeros  
       10-22      Dictionary section numb  
       23-31      Displacement in section

- ③ Bits        Meaning, if on  
       0            Literal  
       1-7          Unused

If bits 0-3 contain any other value, then the configuration is as follows:

- ④ Bits        Contents  
       0-1        Zeros  
       2-14       Dictionary section numb  
       15-23      Displacement in section

<u>Bits</u>	<u>Field</u>	<u>Meaning</u>
0-3	i	Type of BL containing base address of area: 0000 = BL 0001 = BLL 0100 = SBL
4-15	d	Displacement from base address
16-23	k	BL number

XSSNT  
 (TIB 4)

Purpose

Store pointers to XSCRPT table during calculation of subscripted or indexed addresses.

2	2
ID number of subscript or index computation	Displacement in XSCRPT table of new address parameter of subscripted or indexed item

Entry Frequency

One entry for each entry in XSCRPT table.

Phases Involved

Phase 50 builds this table while building XSCRPT table.

Phase 50 uses this table to locate entries in XSCRPT table.

INTERNAL TEXT FORMATS

The types of compiler text produced by each phase are given in Figure 61. In this appendix, there is a separate section describing each type of text. The sections are in the following order:

- IPTEXT
- Data IC-text
- ATF-text
- Data A-text
- Procedure IC-text (P0 Form)
- Procedure IC-text (P1 Form)
- Procedure IC-text (P1A Form)
- Procedure IC-text (P2 Form)
- ATM-text
- Procedure A-text
- Optimization A-text
- Procedure A1-text
- E-text
- XREF-text
- Debug-text

With some exceptions, one IC-text element represents one source element. A source element is a COBOL reserved word, a punctuation symbol, an arithmetic operator, a relational symbol, an EBCDIC name, or a literal. The major exception is that one IC-text element represents a complete data item description. Other exceptions are: the word DIVISION is suppressed in division headers, the word SECTION is suppressed in

section headers, and standard paragraph-names are omitted.

All internal text elements begin with an identifier byte. In IC-text and E-text the first two bits of this byte contain a code with the following significance:

<u>Code</u>	<u>Meaning</u>
01	1 byte follows
10	2 bytes follow
11	3 bytes follow
00	The byte immediately following this gives the number of bytes that follow it.

NOTES ON TEXT ELEMENT FORMATS

- The top row of figures shows the byte numbers occupied by each field, except where the field is preceded by a variable-length field.
- **Boxes that are shaded** define optional fields or a series of similar fields.
- c = the number of bytes in the field that follows.
- n = the total number of bytes that follow in the remainder of the text element.
- 1b = this field is one byte long.
- s = size, in words, of the block section or area to which the element refers.
- Pairs of characters in byte 0 are hexadecimal numbers.

| IPTEXT

| BASIC LISTER FORMAT

1	1	1	Variable
type code	card	length	text
①	column number	②	

| SYNTACTIC AND REFERENCE MARKERS (ONE BYTE)

1
type code
③

| SYNTACTIC AND REFERENCE MARKERS (TWO BYTE)

1	1
type code	modifier code
④	

① Type Code (Hex)	Meaning
50	A COBOL source name (subject to cross-referencing)
51	A COBOL reserved word
52	Syntactic string with no special lister importance (for example, text from a note paragraph)
53	Left punctuation (printed on left end of string with no intervening blanks)
54	Right punctuation (printed on right end of string with no intervening blanks)
55	A prose element from an *-comment card
58	Initial ETEXT elements (with card numbers)
59	Subsequent ETEXT elements (without card numbers)

② Length in bytes of the text in the following variable field.

③ Type Code (Hex)	Meaning/Usage
00	Marks beginning of a COBOL source Definition (range is 0-51 and denotes the nesting depth)
33	Marks beginning of a COBOL source
34	Statement in column 8, 10 or margin B
35	Column 12 or greater,

3C	respectively Marks beginning of a COPY statement
3D	Marks end of a COPY statement
3E	Marks beginning of a copied library member
3F	Marks end of a copied library member
36	Marks beginning of a reference to a source definition
3A	Marks beginning of a reference to the (TALLY, TALLYING) register
37	Indent to the left two places
38	Indent back to the right two places
39	Cancel all outstanding BCLSEs.

④ Type Code (Hex)	Meaning
45	Indicates a COBOL source clause and the modifier code indicates where this clause belongs in a standard ordering. In phase 08, a COBOL source statement continuing several BCLSn - clauses will have these sorted in ascending sequence on the modifier code value prior to printing and punching of that source statement.
44	Modifier = C'E'. Marks beginning of erroneous, missing, or out of order COBOL source text.



44 Modifier = C'R'. Marks end of erroneous COBOL source text.

The following codes have a four-byte format, and are used by phase 06 for creating and imbedding cross-reference information into the IPTEXT which is passed to phase 08:

Code	<u>Meaning/Usage</u>
<u>Hex</u> 48	Marks end of a source DEFN reference (a source name followed by zero or more qualifiers). Phase 05 creates this and phase 06 fills in the statement field.
49	Marks end of a reference to a Procedure Division definition.

The following codes are created by phase 06:

Code	<u>Meaning/Usage</u>
<u>Hex</u> 4A	Information collected at the point of definition indicating the type and statement number of the reference.

4B Information collected at the point of definition indicating the type and statement number of the reference. May be moved to another definition prior to final output to phase 08. (Note: IPTEXT information passed to phase 08 will contain no CONDREFS.)

4C Tells phase 06 the statement number of the Procedure Division section from which the following CONDREFS came.

The format for X'48' through X'4C' is as follows:

1	1	2
type	reference	Reference/definition
code	type	statement number

DATA IC-TEXT

LD ENTRY

0	1	2	3-5	6	7	8-9	10	11
03	n	Level indicator	Source card number	Switch byte	Switch byte	OCCURS DEPENDING ON maximum occurrences	Switch byte	Number of indexes following
		(1)		(2)	(3)		(4)	

12	13	Variable	1b	Variable	1b	Variable	1b	Variable	Variable
Number of keys following	c	Name of data item	c	PICTURE (actual)	c	Encoded VALUE	c	REDEFINES data-name	OCCURS DEPENDING ON pointer
				(5)		(6)		(7)	(8)

Note: A series of logical records can follow the LD entry. The types of records are ordered as follows:

Value for condition-name with multiple values	1b	1b	Variable
	Switch byte	(9)	c
			Encoded VALUE

Indexes (first) and/or Keys (second)	1b	1b	Variable
	Flag byte	(10)	c
			Index-name or key-name in EBCDIC

Object of RENAMES or RENAMES THRU	1b	1b	Variable
	ID code	(11)	c
			Name in EBCDIC

SD ENTRY

0	1	2	3	4-6	7-8	9-10
03	n	Level indicator 36 (hex)	(12)	Source card number	Minimum RECORD CONTAINS value	Maximum RECORD CONTAINS value

11	12	13-20	21	22-51
(13)	SAME RECORD AREA number	Unused	c	Sort-name in EBCDIC; low-order unused bytes padded with blanks

RD ENTRY

0	1	2	3	Variable	1b
03	n	Level indicator 34 (hex)	c	User-assigned EBCDIC report-name	00

FD ENTRY

0	1	2	3-5	6-13	14	15	16	17	18-19
03	n	Level indicator 38 (hex)	Source card number	(14)	Switch byte	Switch byte	Switch byte	Buffer offset value	Displacement of entry for file in PIOTBL
				(15)	(16)	(28)			

20	21	22-23	24-25	26-27	28-29
Number of areas specified in RESERVE clause	Switch byte (17)	Integer-1 specified in BLOCK CONTAINS	Minimum number of record bytes	Maximum number of record bytes	Number of TRACK LIMIT tracks

30-31	32-33	34	35	36	37	38	39-42	43
Unused	Displacement of entry for file in CKPTBL	SAME AREA number	Switch byte (18)	Switch byte (19)	Switch byte (20)	SAME RECORD AREA number	Unused	Unused

44-45	46	47	48-49	50	51	52	53-54	55
BLOCK CONTAINS maximum count	Switch byte (21)	Count of text elements following IDCNT	Displacement of IDs in IND_TBL	Unused	LINAGE	Count of PASSWORDS for ALTERNATE KEY	Unused	c
					(29)			

Variable	Variable	Variable	Variable	Variable	Variable
File-name in FD entry	TRACK AREA size	NOMINAL KEY and qualifiers	ACTUAL KEY and qualifiers	RECORD KEY and qualifiers	REORG-CRITERIA data-name and qualifiers
	(22)	(23)	(23)	(23)	(23)

Variable	Variable	Variable	Variable	Variable	Variable
FILE STATUS data-name and qualifiers	PASSWORD data-name and qualifiers	RELATIVE KEY and qualifiers	TOTALING AREA data-name and qualifiers	LINAGE data-name or integer	FOOTING data-name or integer
(23)	(23)	(23)	(23)	(30)	(30)

Variable	Variable	Variable	1B
TOP data-name or integer	BOTTOM data-name or integer	LABEL RECORDS names	00
(30)	(30)	(24)	

ID ENTRY

0	1	2	3	4-6
IC types	Count of fixed position	IC level	Flag byte	Card number
'03'	'05'	(31)	(32)	

CD ENTRY

0	1	2	3-5	6	Variable
03	n	Level indicator (hex)	Source card number	Switch byte	CD-name in EBCDIC
		(37)		(25)	

Note: If bit 0 is set to 0 in the Switch byte, the following fields are also generated.

Variable	Variable	Variable	Variable	Variable	Variable	Variable	Variable	Variable
Data-name-1	Data-name-2	Data-name-3	Data-name-4	Data-name-5	Data-name-6	Data-name-7	Data-name-8	Data-name-11
(26)	(26)	(26)	(26)	(26)	(26)	(26)	(26)	(26)

Note: If bit 0 is set to 1 in the Switch byte, the following fields are also generated.

Variable	Variable	Variable	Variable	Variable	Variable
Data-name-1	Data-name-2	Data-name-3		Data-name-4	Data-name-5
(26)	(26)	(26)	(28)	(26)	(26)

CRITICAL PROGRAM BREAK

0	1
42	Type of break
	(27)

(28) DESTINATION TABLE ENTRY:

0-1	2	Variable	Variable	...	Variable
number of occurrences	number of index entries	index-name-1	index-name-2		index-name-n
		(29)			(29)

(29) INDEX-NAME FORMAT:

0	1	Variable
Prefix		EBCDIC for index-name-n
04	C	

①	<u>Code (hex)</u>	<u>Meaning</u>
	01-31	= Levels 01-49
	32	= Level 77
	33	= Level 88
	34	= RD
	36	= SD
	37	= CD
	38	= FD, or in Report Section, = Special RD elements
	39	= Level 66

101 = SIGN is separate trailing  
111 = SIGN is separate leading  
5 If there is an OCCURS  
6-7 Unused

②	<u>Bits</u>	<u>Code</u>
	0	1 = BLANK WHEN ZERO
	1	1 = JUSTIFIED
	2-4	<u>Type of VALUE Clause</u>
		000 = No clause
		001 = Alphanumeric literal
		010 = Numeric literal
		011 = Floating-point literal
		100 = Figurative constant or ALL
		101 = Figurative constant ZERO
		111 = Condition-name with multiple values
	5-7	<u>Type of USAGE</u>
		000 = No clause
		001 = DISPLAY
		010 = COMPUTATIONAL
		011 = COMPUTATIONAL-1
		100 = COMPUTATIONAL-2
		101 = COMPUTATIONAL-3
		110 = DISPLAY-ST
		111 = INDEX

- ⑤ Contains zeros if this value is a condition-name with multiple values. These values follow the LD entry.
- ⑥ VALUE encoded like a figurative constant, literal, or ALL character in Procedure IC-text, except: for numeric literal, digits not packed but in EBCDIC format, with sign in zone of low-order digit.

Note: This field contains zeros when the item is a condition-name that is part of a VALUE...THRU...clause.

③	<u>Bits</u>	<u>Code</u>
	0	1 = OCCURS DEPENDING ON
	1	1 = REDEFINES
	2	1 = PICTURE
	3	1 = COPY
	4	1 = Internal REDEFINES (RD entry)
	5	1 = S.nnnn description and therefore the PICTURE field contains the E.nnnn name from which the PICTURE information is to be extracted.
	6	1 = RENAMES data-name follows
	7	1 = SYNCHRONIZED

- ⑦ Zero if no REDEFINES clause.
- ⑧ a. If there is an OCCURS DEPENDING ON clause, the field is a 16-bit number representing displacement from start of OD2TBL of entry for object of clause.
- b. If internal REDEFINES (RD entry), the field is a 16-bit number representing the displacement which added to the object gives address of REDEFINES subject.
- c. If neither, field is eight bits of zeros.

④	<u>Bits</u>	<u>Meaning, if SYNCHRONIZED</u>
	0	0 = SYNC LEFT 1 = SYNC RIGHT
	1	RENAMES THRU data-name follows
	2-4	001 = SIGN is overpunch trailing 011 = SIGN is overpunch leading

⑨	<u>Bits</u>	<u>Contents</u>
	0-2	Zeros
	3	1
	4	0 = Either value is upper limit of THRU range, or THRU was not specified. 1 = Value is lower limit of range; upper limit name follows.
	5-7	<u>Value</u> <u>Meaning</u>
		001    Alphanumeric literal
		010    Numeric literal
		011    Floating-point literal
		100    Figurative constant or ALL
		101    Figurative constant ZERO

⑩	<u>Bits</u>	<u>Meaning</u>
	0-3	Zeros
	4	Unused
	5	1 = INDEXED BY
	6	1 = DESCENDING KEY
	7	1 = ASCENDING KEY

⑪	<u>Code</u>	<u>Meaning</u>
	22	This name qualifies the name that follows.
	23	This is either a name without qualifiers, or it is the last (qualified) name in a string.

⑫	<u>Bits</u>	<u>Contents</u>
	0-5	Unused
	6	1 = SAME RECORD AREA
	7	Unused

⑬	<u>Bits</u>	<u>Contents</u>
	0-2	Unused
	3-6	<u>Code</u> <u>Record Format</u>
		1000        F
		0100        V
		0010        U (invalid)
		0001        S
	7	1 = ASCII collating sequence

⑭ ddname portion of system-name of file, padded with blanks, if necessary.

⑮	<u>Bits</u>	<u>Code</u>
	0	1 = RANDOM ACCESS
	1-3	<u>Code</u> <u>Organization</u>
		000 SEQUENTIAL 'S'
		001 INDEXED
		010 DIRECT with REWRITE 'W'
		011 DIRECT 'D'
		100 RELATIVE
	4-6	Reserved
	7	1 = RESERVE ALTERNATE AREA

⑯	<u>Bits</u>	<u>Meaning, if on</u>
	0	OPTIONAL in SELECT
	1	SAME AREA
	2	Unused
	3	SAME RECORD AREA
	4	SAME SORT AREA
	5	This entry contains pointer to CKPTBL
	6	This entry contains pointer to PIOTBL
	7	Word 'ALTERNATE' specified in RESERVE clause

⑰	<u>Bits</u>	<u>Code</u>
	0	1 = COPY
	1	Unused
	2	RECORD CONTAINS CHARACTERS
	3-4	BLOCK CONTAINS option
		00 = Not specified
		01 = RECORDS
		10 = CHARACTERS
	5-6	LABEL RECORDS option
		00 = Not specified
		01 = STANDARD
		10 = OMITTED
		11 = Data-name
	7	1 = REPORT clause

⑱	<u>Bits</u>	<u>Code</u>
	0-1	TRACK AREA
		00 = Not specified
		01 = Data-name
		10 = Integer
	2	1 = RELATIVE KEY
	3	1 = NOMINAL KEY
	4	1 = ACTUAL KEY
	5	1 = RECORD KEY
	6	1 = WRITE ONLY
	7	FILE STATUS specified under SELECT

⑲	<u>Bits</u>	<u>Meaning, if on</u>
	0-6	Unused (4 multiple files)
		RESERVE integer not in valid range
	7	RECORD OVERFLOW

⑳	<u>Bits</u>	<u>Meaning, if on</u>
	0-3	Unused
	4	CORE-INDEX
	5	REORG-CRITERIA
	6	ASCII file
	7	Unused

Bits	Meaning, if on
0	TOTALING AREA
1	TOTALED AREA (Unused after phase 1B)
2	Format F
3	Format V RECORDING MODE
4	Format U
5	Format S
6-7	Unused

(22) Either name containing size of TRACK AREA preceded by count-byte of character or 2-byte field giving integer TRACK AREA count.

Subfield	Contents
1	2-byte count of bytes in all the subfields that follow in this field.
2	Name of highest level qualifier preceded by 1-byte count of characters.
.	.
.	.
n	Name of lowest-level qualifier preceded by 1-byte count of characters.
n+1	Zero, to separate this field from the next. If the option is not specified, the field consists of one byte of zeros.

(24) Series of all label record-names, each preceded by a 1-byte count of characters.

Bit	Meaning
0	1 = OUTPUT
1	1 = Variable entries follow
2	1 = CD for INITIAL INPUT
3	1 = Destination table specified

Byte	Meaning
0	Count of bytes in the field that follows (c)
1-c	EBCDIC name of data-name-n

If a data-name is not specified, the field will consist of one byte of zeros.

Code (hex)	Type of Break
01	Data Division
02	File Section
03	Working-Storage Section
04	Linkage Section
05	Report Section
06	Procedure Division
0E	Communication Section

FSEVAM	
Bit	Meaning
0	1 = 'AS' specified as organization parameter
1	1 = no organization parameter
2-4	ORGANIZATION clause
2	RELATIVE
3	INDEXED
4	SEQUENTIAL
5	ACCESS MODE DYNAMIC
6	ALTERNATE RECORD KEY(S)
7	PASSWORD specified

LINESW	
Bit	Meaning
0	LINAGE clause specified
1	FOOTING option specified
2	TOP option specified
3	BOTTOM option specified
4	Object by LINAGE is data-name
5	Object by FOOTING is data-name
6	Object by TOP is data-name
7	Object by BOTTOM is data-name

(30) If the field is a data-name, it will be preceded by a one-byte count field. If integer, the field will be eight bytes in length. If LINAGE clause or other three clauses are not specified, each unspecified field will be replaced by one byte of zero.

Code	Meaning
35	Level is ID

Bit	Meaning
0-2	Unused
3	1 = Another ID follows
4	1 = PASSWORD specified
5	1 = DUPLICATES specified
6-7	Unused

ATF-TEXT

Level	0	1	2	3-5	6-7	8-9	10	11	12-13
01-49 or 77 items	03	n	Level number	Card number	Flag	Maximum number of occurrences	Number of indexes	Number of keys	Length of the item in the object module
			①			②			

14-Variable	Variable	Variable	Variable	Variable
EBCDIC name of item	EBCDIC name of object	Table displacement	Partial dictionary attributes	Table displacement
③	④	⑤	⑥	⑦

Level	0	1	2	3-5	Variable	Variable
88 items	03	n	X'33' (Level number)	Card number	EBCDIC name of item	Partial dictionary attributes
					③	⑥

① The maximum length of any element is 204 bytes.

② The flag indicates the origin of the element, as follows:

Bit	Meaning
0	RENAMES...THRU clause
1	Next element is an FD
2	Next element is an SD
3	Next element is an RD
4	Conditional variable
5	Data A-text follows
6	VALTRU table entry
7	VALGRP table entry
8	ODO
9	REDEFINES clause
10	USAGE is not DISPLAY
11	Item is or is in a LABEL record.
12	Internal redefines
13	RD
14	RENAMES clause
15	SYNCHRONIZED.

③ The name is prefixed by a 1-byte count of its length.

④ Either:

1. A 1-byte length count followed by the objects of the REDEFINES clause, if flag bit 9 is on; or

2. A 1-byte length count followed by the object of the internal REDEFINES clause, if flag bit 12 is on; or
3. A 1-byte length count followed by the Report Section (RD) name, if flag bit 13 is on; or
4. The field does not exist.

⑤ Either:

1. A 2-byte OD2TBL table displacement if flag bit 8 is on; or
2. A 2-byte internal redefines displacement, if flag bit 12 is on; or
3. The field does not exist.

⑥ The attributes are prefixed by a 1-byte count of their length

⑦ Either:

1. A 2-byte VALGRP table displacement if flag bit 7 is on; or
2. A 2-byte VALTRU table displacement if flag bit 6 is on; or
3. A 2-byte VALGRP table displacement followed by a 2-byte VALTRU table displacement if flag bits 6 and 7 are on; or
4. The field does not exist.



DATA A-TEXT

DCB ADDRESS

0	1-3	4		
04	Relative address in object module of DCB	Number assigned from DCBCTR in COMMON		

DECB ADDRESS

0	1-3	4	5-6	7-8
08	Relative address in object module of DECB	Number assigned from DECBCT in COMMON	00	s

BLOCK ADDRESS

0	1-3	4	5-6	7-8
0C	Relative address pointed to by base locator described in next field	BL number -- first base locator number assigned to file from BLCTR in COMMON	00	s

FIB ADDRESS

0	1-3	4		
14	Relative address in object module of File Information Block	FIB number -- File Information Block number assigned from AMICTR in COMMON		

COUNT INFORMATION

0	1-3	4	5-6	7 through 6 + c
20	Relative address following Q-routines during Data-A-text processing	00	c	Actual constant (COUNT table information)

WORKING-STORAGE SECTION ADDRESS

0	1-3	4	5-7	
24	Relative address in object module of Working-Storage Section	BL number -- first base locator number assigned to Working-Storage Section from BLCTR in COMMON		s

CONSTANT DEFINITION

0	1-3	4	5-6	7 through 6 + c
28	Relative address in object module where constant is located	Type of constant	c	Actual constant. If ALL, occupies only first byte.
		①	②	②

ADDRESS CONSTANT DEFINITION

0	1-3	4	5-7
2C	Relative address in object module where address constant is located	Size, in bytes, of address constant	Relative address in object module specified by address constant

DELIMITER

0
01 (3)

Q-ROUTINE IDENTIFICATION

0	1-2
34	GN number -- generated procedure-name number assigned from GNCTR in COMMON

BL REFERENCE

0	1-3	4
38	Relative address from beginning of the program where displacement for base locator cell described in next field is to be placed	BL number -- base locator number

BLL REFERENCE

0	1-3	4
3C	Relative address from beginning of the program where displacement for base locator cell described in next field is to be placed	BLL number -- base locator number

①	Code (hex) <u>Meaning</u>
	00    Binary
	01    Alphanumeric
	FF    ALL constant or figurative constant

② If the constant is an ALL constant, the format differs beginning with byte 5 as follows, where d is the number of bytes reserved for the constant:

5-6	7	8 through 7 + c
d	c	Value specified for the constant

③ This element is written on SYSUT4 by phase 51 if the SYMDMP or the STATE and OPT options are in effect. It identifies the end of Data A-text, DEF-text, and E-text and the beginning of Debug-text, which phase 63 writes on SYSUT4 for use by phase 65.

PROCEDURE IC-TEXT (PO FORMAT)

PROCEDURE-NAME DEFINITION

0	1	2 through 1 + c
05	c	Procedure-name in EBCDIC

QUALIFYING EBCDIC NAME

0	1	2 through 1 + c
22	c	User-assigned name in EBCDIC that qualifies procedure-name or data-name

EBCDIC NAME

0	1	2 through 1 + c
23	c	User-assigned name in EBCDIC

REPORT WRITER RECORD CONTAINING CODE OPTION

0	1	2-6
24	05	60F0F0F0F1 or F2

EBCDIC DATA-NAME OF GIVING OPTION FOR USE ERROR DECLARATIVE

0	1	2 through 1 + c
25	c	Use assigned EBCDIC name that was object of GIVING option

GN'S FOR ERROR/LABEL DECLARATIVES

0	1	2-3	4-5
26	0A	GN number for STANDARD ERROR (1)	GN number for file header labels (1)

6-7	8-9
GN number for file trailer labels (1)	GN number for end-of-volume labels (1)

10-11
GN number for beginning-of-volume labels (1)

NUMERIC LITERAL

0	1	2	3
32	n	Positions to left of decimal	Positions to right of decimal

4 through 1 + n
Literal in packed decimal format

FLOATING-POINT LITERAL

0	1	2-9
33	08	Literal represented as double-precision floating-point number

ALPHANUMERIC LITERAL

0	1	2 through 1 + c
34	c	Literal in EBCDIC

"EXHIBIT NAMED" NAME

0	1	2 through 1 + c
35	c	EBCDIC name used in EXHIBIT NAMED statement

LISTING A-TEXT FOR PROCEDURE-NAMES

0	1	2 through 1 + c
37	c	EBCDIC procedure-name bit 0 of preceding field is set to 1

LISTING A-TEXT FOR VERBS

0	1	2 through n	n + 1
37	n	EBCDIC verb	Alphabetic verb sequence number

CRITICAL PROGRAM BREAK

0	1
42	Break code (2)

VERB

0	1
44	Verb code (3)

RELATIONAL CODE

0	1
50	06 (hex) = Equal
	08 = Greater than
	0A = Less than
	0C = Not equal

PARENTHESIS

0	1
52	00 (hex) = Left parenthesis
	01 = Right parenthesis

ARITHMETIC OPERATOR

0	1
53	Operator code (4)

COBOL WORD

0	1	2
54	Word code	Code (Phases 10, 12, and 1B only, not passed on)
	(7)	(8)

SPECIAL NAME

0	1
55	Code (5)

COBOL WORD 2

0	1	2
57	Word code	
	(9)	(8)

FIGURATIVE CONSTANT

0	1
75	EBCDIC value of figurative constant

STANDARD DATA-NAME REFERENCE

0	1
79	01 = LINE-COUNTER
	02 = PAGE-COUNTER
	05 = TALLY

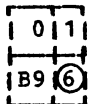
GENERATED PROCEDURE-NAME DEFINITION

0	1-2
88	GN number -- identifying number assigned to compiler-generated procedure-names from COMMON field GNCTR

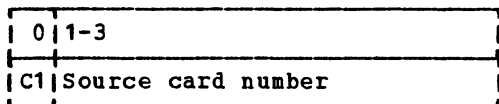
GENERATED PROCEDURE-NAME REFERENCE

0	1-2
AA	GN number -- identifying number assigned to compiler-generated procedure-names from COMMON field GNCTR

ERROR SYMBOL



CARD NUMBER



① If there is no GN for this purpose, the field contains zeros.

②

Code (hex)	Meaning
01	Data Division
02	File Section
03	Working-Storage Section
04	Linkage Section
05	Report Section
06	Procedure Division
07	Start of declaratives
08	End of declaratives
09	Start of debug packets
0A	Start of Q-Routines
0B	Start of Report Writer procedures
0C	End of Report Writer procedures
0D	End of segment
0E	Communication Section
0F	Date-Compiled entry
F0	Security entry
F1	Identification Division
F2	Program-ID entry
F3	Author entry
F4	Environment Division
F5	Configuration Section
F6	Source-Computer entry
F7	Object-Computer entry
F8	Input-Output Section
F9	File-Control entry

③ Verb Code List: Code indicates the type of verb.

Code	Meaning	
	P0- and P1-text	P2-text
00	ADD	ADD
01	SUBTRACT	SUBTRACT
02	MULTIPLY	MULTIPLY
03	DIVIDE	DIVIDE
04	COMPUTE	EXPONENTIATE
05		STORE
06	END OF SENTENCE	IF-EQ-NUMERIC
07	IF	IF-NOTEQ-NUMERIC
08	ELSE (OTHERWISE)	IF-GT-NUMERIC
09		IF-NOTGT-NUMERIC
0A		IF-LT-NUMERIC
0B		IF-NOTLT-NUMERIC
0C		IF-ALPHABETIC
0D		IF-NOT-ALPHABETIC
0E		IF-NUMERIC
0F		IF-NOT-NUMERIC
10	STOP	STOP
11	GO	GO
12		GO-DEPEND-FIRST
13		GO-DEPEND-MIDDLE
14		GO-DEPEND-LAST
15		EVAL
16		IF-EQ-NONNUM
17		IF-NOTEQ-NONNUM
18		IF-GT-NONNUM
19		IF-NOTGT-NONNUM
1A		IF-LT-NONNUM
1B		IF-NOTLT-NONNUM
1C	ALTER	MOVE-4
1D	MOVE	MOVE
1E	EXAMINE	EXAMINE
1F	TRANSFORM	TRANSFORM
20	READ	READ
21	OPEN	OPEN
22	CLOSE	CLOSE
23	WRITE	WRITE
24	REWRITE	REWRITE
25	ACCEPT	ACCEPT
26	DISPLAY	DISPLAY
27	EXHIBIT	EXHIBIT
28	RESET	RESET

Code	Meaning					
	P0- and P1-text	P2-text				
29	READY	READY		57	REPORT-RETURN-1	REPORT-RETURN-1
2A	RETURN	RETURN		58	REPORT-RETURN-2	REPORT-RETURN-2
2B	ON	ON		59	REPORT-RETURN-3	REPORT-RETURN-3
2C	ENTRY	ENTRY		5A	REPORT-RETURN-4	REPORT-RETURN-4
2D	CALL	CALL		5B	REPORT-RETURN-5	REPORT-RETURN-5
2E	ENTER			5C	REPORT-ORIGIN	REPORT-ORIGIN
30	CANCEL	CANCEL		5D	REPORT-REORIGIN	REPORT-REORIGIN
31	USE (except UFD)			5E	SEARCH	Beginning of WHEN in SEARCH ALL
32	EXIT			5F	SEARCH ALL	End of WHEN in SEARCH ALL
33	REPORT-NOP			60		SET format-1
34	GENERATE			61	SET	SET format-2 (UP BY)
35	TERMINATE			62	See note following list	
36	SORT	SORT		63	SEEK	
37	RELEASE	RELEASE		64	START	START
38	PERFORM	GO-N-TIMES		65	UNSTRING	UNSTRING
39		SUBSCRIPT		66		IF EQUAL (index-name)
3A	INITIATE			67		IF NOT EQUAL (index-name)
3B	DEBUG (ON)	DEBUG		68		IF GREATER (index-name)
3C		START KEY		69		IF NOT GREATER (index-name)
3D		TRACE		6A		IF LESS (index-name)
3E		EQUATE		6B		IF NOT LESS (index-name)
3F		MOVE-1 (Report Writer verb)		6D		EQUATE in SEARCH ALL
40	INIT	INIT		6E		SET format-2 (DOWN BY)
41	INCRA	INCRA		6F		GO TO (Segmentation)
42	STEP	STEP		70		Segmentation initialization verb
43	UPDATE	UPDATE		71	READ (for RERUN file)	READ (for RERUN file)
44		USE-ERROR		72	WRITE (for RERUN file)	WRITE (for RERUN file)
45		ENDUSE-ERROR		73	GOBACK	GOBACK
46		USE-LABELS		74		EXIT program
47		ENDUSE-LABELS		75	STRING	STRING
48		ACCEPT MESSAGE				
4A		USE-REPORT				
4B		ENDUSE-REPORT				
4C	Q-CALL	Q-CALL				
4D	Q-RETURN2	Q-RETURN2				
4E	Q-RETURN3	Q-RETURN3				
4F	REPORT-CALL	REPORT-CALL				
50	REPORT-SAVE-0	REPORT-SAVE-0				
51	REPORT-SAVE-1	REPORT-SAVE-1				
52	REPORT-SAVE-2	REPORT-SAVE-2				
53	REPORT-SAVE-3	REPORT-SAVE-3				
54	REPORT-SAVE-4	REPORT-SAVE-4				
55	REPORT-SAVE-5	REPORT-SAVE-5				
56	REPORT-RETURN-0	REPORT-RETURN-0				

<u>Code</u>	<u>Meaning</u>	
	<u>P0- and P1-text</u>	<u>P2-text</u>
76	SETVLC (for RENAME Q-Routine)	SETVLC (for RENAME Q-Routine)
77		Flow trace (for source PN's only)
78	RECEIVE	RECEIVE
79		OPEN (VSAM)
7A		CLOSE (VSAM)
7B		Subroutine test (for IF MESSAGE, STRING, and UNSTRING)
7C	GNRPT (for OPT)	GNRPT (for OPT)
7D	SEND	SEND
7E		READ (VSAM)
7F		WRITE (VSAM)
80		REWRITE (VSAM)
81		START (VSAM)
82	DELETE (VSAM)	
83	DEBUG transfer of control	DEBUG transfer of control
84		QCALL2 (for UNSTRING)
85		UNSTRING header (PH45)
87	MERGE	MERGE
88		COUNT
8A		UFD Debug verb
8D	INSPECT	
90	USE FOR DEBUGGING	
91	ENABLE	
92	DISABLE	
93		ACCEPT MESSAGE
94		End-of-UFD section
95		RFRSEG
96		Debug subscript verb

Note: Code 62 indicates SEGMENT-LIMIT and is used for E-text only.

④

<u>Code</u> <u>(hex)</u>	<u>Operator</u>
00	Addition
01	Subtraction
02	Multiplication
03	Division
04	Exponentiation

⑤

<u>Code</u> <u>(hex)</u>	<u>Special</u> <u>Name</u>
00	CSP
01	C01
02	C02
03	C03
04	C04
05	C05
06	C06
07	C07
08	C08
09	C09
0A	C10
0B	C11
0C	C12
0D	S01
0E	S02
0F	S03
10	S04
11	S05
12	PAGE

⑥

<u>Error Symbol</u> <u>COBOL word</u> <u>code</u>	<u>When Used</u>
C0 (hex)	Reserved word used invalidly. Undefined or multiply-defined symbol found.

⑦ COBOL Word Code: Number assigned to identify a COBOL word. Note that in phases 10, 12, and 1B of the compiler listing, these words appear in alphabetical order according to length of the word in the COBOL word table (COBWRD).

<u>Code</u>	<u>Word</u>
01	DATA
02	SKIP1
03	SKIP2
04	SKIP3
05	EJECT
06	NSTD-REELS
07	SUPPRESS
0A	ORGANIZATION
0C	CORE-INDEX
0D	PROGRAM
0E	RF
0F	WRITE-ONLY
10	TOTALING
11	TOTALED
12	COMMA
13	DECIMAL-POINT
14	FILE-LIMIT(S)
15	MODE
16	RECORDING
17	REEL
18	SYSIN

<u>Code</u>	<u>Word</u>	48	SORT-RETURN
19	SYSOUT	49	SEPARATE
1A	TRACK-AREA	4A	LEAVE
1B	MESSAGE	4B	REREAD
1C	TRACK-LIMIT	4C	DISP
1D	DELIMITED	4D	EXTENDED-SEARCH
1E	POINTER	4E	MASTER-INDEX
1F	OVERFLOW	4F	CYL-OVERFLOW
20	COUNT	50	THEN
21	DELIMITER	51	CYL-INDEX
22	TIME	52	WRITE-VERIFY
23	EGI	53	THAN
24	DATE	54	RECORD-OVERFLOW
25	REORG-CRITERIA	55	ALPHABETIC
26	DISPLAY	56	NUMERIC
28	RESET	57	POSITIVE
29	SEGMENT	58	NEGATIVE
2A	SUB-QUEUE-1	5A	END-OF-PAGE (EOP)
2B	ON	5B	CHARACTER
2C	SUB-QUEUE-2	5C	NOT
2D	SUB-QUEUE-3	5D	AND
2E	INITIAL	5E	OR
2F	SYMBOLIC	5F	LIMIT (S)
30	CURRENCY	60	TEXT
31	QUEUE	61	BEGINNING
32	INDEX	62	ENDING
33	STATUS	63	MORE-LABELS
34	MODULES	64	OUTPUT
35	MEMORY	65	LENGTH
36	WORDS	66	INPUT
37	SYNCHRONIZED (SYNC)	67	RANDOM
38	OFF	68	PROCESSING
39	RENAMES	69	BEFORE
3A	UP	6A	REPORTING
3B	DOWN	6B	I-O
3C	FILE (in Procedure Division and after File Section header)	6C	WITH
3D	OPTIONAL	6D	REWIND
3E	REMAINDER	6E	REVERSED
3F	POSITION	6F	INTO
40	TAPE	70	AT
41	TRAILING	71	INVALID
42	ADDRESS	72	AFTER
43	ALPHANUMERIC	73	ADVANCING
44	NUMBER	74	CBL
45	CURRENT-DATE	75	DEPTH
46	TIME-OF-DAY	76	LOCK
47	TERMINAL	77	SYSPUNCH
		78	CONSOLE



<u>Code</u>	<u>Word</u>		
79	ALL	A5	SECTION
7A	CORRESPONDING (CORR)	A6	LABEL-RETURN
7B	TALLYING	A7	DIVISION
7C	LEADING	A8	SORT-FILE-SIZE
7D	UNTIL	A9	SORT-CORE-SIZE
7E	REPLACING	AA	SORT-MODE-SIZE
7F	BY	AB	SIGN
80	DESTINATION	AC	SORT (appears in Procedure Division as verb with 36 code)
81	GIVING	AD	MULTIPLE
82	ROUNDED	AE	EXCEPTION
83	SIZE	AF	FILLER
84	ERROR	B0	ESI
85	RUN	B1	ASSIGN
86	PROCEED	B2	ACCESS
87	THROUGH (THRU)	B3	EMI
88	VARYING	B4	RESERVE
89	USING	B5	NOMINAL
8A	COBOL	B6	ACTUAL
8B	DAY	B7	TABLE
8C	DESCENDING	B8	DYNAMIC
8D	ASCENDING	B9	Reserved
8E	TRACE	BA	SEQUENTIAL
8F	CHANGED	BB	DEBUGGING
90	NAMED	BC	INDEXED
91	LINKAGE	BD	SORT-MERGE
92	CHARACTER(S)	BE	ALTERNATE
93	TIMES	BF	AREA(S)
94	DEPENDING	C0	SORT-MESSAGE
95	LINE(S)	C1	RELOAD
96	FIRST	C2	RELATIVE
97	NEXT	C3	SEARCH
98	UPON	C4	TRACK(S)
99	PROCEDURE (in Procedure Division)	C6	PASSWORD
9A	EVERY	C7	PROTECTION
9B	TO	C8	LIBRARY
9C	IS, ARE	C9	EXTEND
9D	FROM	CA	VALUE(S)
9E	NO	CB	PRINT-SWITCH
9F	KEY	CC	BLOCK
A0	RETURN-CODE	CD	RECORD
A1	END	CF	RECORDS
A2	UNIT(S)	D0	CONTROL(S)
A3	FOR	D1	LABEL(S)
A4	IN, OF	D3	CONTAINS
		D4	OMITTED
		D5	STANDARD
		D6	REPORT(S)
		D7	REDEFINES
		D8	PICTURE (PIC)
		D9	BLANK

<u>Code</u>	<u>Word</u>		
DA	OCCURS	4	Allowed in Environment Division
DB	JUSTIFIED (JUST)	5	Allowed in Data Division
DC	POSITIONING	6	Allowed in Procedure Division
DD	USAGE	7	Allowed in Identification Division
DE	COMPUTATIONAL (COMP)		
	COMPUTATIONAL-4 (COMP-4)		
DF	COMPUTATIONAL-1 (COMP-1)		
E0	COMPUTATIONAL-2 (COMP-2)		
E1	COMPUTATIONAL-3 (COMP-3)		
E2	WHEN		
E3	RIGHT		
E4	LEFT		
E5	CODE		
E6	PAGE		
E7	FINAL		
E8	REMOVAL		
E9	HEADING		
EA	DETAIL (DE)		
EB	LAST		
EC	FOOTING		
ED	UPSI-0 through UPSI-7		
EE	GROUP		
EF	TYPE		
F0	PLUS		
F1	LINAGE		
F2	DISPLAY-ST		
F3	RH		
F4	PH		
F5	BOTTOM		
F6	CH		
F7	NOTE		
F8	CF		
F9	TOP		
FA	PF		
FB	SENTENCE		
FC	COLUMN		
FD	INDICATE		
FE	SOURCE		
FF	SUM		

9		<u>Code</u>	<u>Word</u>
	COBOL	WORD 2	
		01	
		02	BASIS
		03	ALSO
		04	REFERENCES
		05	PROCEDURES
		06	COLLATING
		07	SEQUENCE
		08	STANDARD-1
		09	NATIVE
		0A	CODE-SET
		0B	DUPLICATES
		0C	INSERT
		0D	DEBUG-ITEM
		0E	DEBUG-LINE
		0F	DEBUG-NAME
		10	DEBUG-SUB-1
		11	DEBUG-SUB-2
		12	DEBUG-SUB-3
		13	DEBUG-CONTENTS

8	<u>Bits</u>	<u>Meaning</u>
	0	FD, SD, RD
	1	Paragraph word
	2	Section word
	3	Division word

PROCEDURE IC-TEXT (P1 FORMAT)

PROCEDURE-NAME DEFINITION

0	1	2 through 1 + c	2 + c
06	c	Dictionary attributes of procedure-name (see "Section 5. Data Areas") ①	Priority number if not a section-name

n - 7 to n - 6	n - 5 to n - 4
PN number for file trailer label ②	PN number for end-of-volume label ②

n - 3 to n - 2	n - 1 to n + 1
PN number for beginning-of-volume label ②	Pointer to dictionary entry for file ③

PROCEDURE-NAME REFERENCE

0	1	2 through 1 + c	2 + c
20	c	Dictionary attributes of procedure-name (see "Section 5. Data Areas") ①	Priority number if not a section-name

SD ELEMENT

0	1	2-9
21	n	Dictionary attributes for SD (see "SD ENTRY" in "Section 5. Data Areas")

FILE-NAME REFERENCE

0	1	variable (9 to 16 bytes)
21	n	Dictionary attributes of file (see "FD ENTRY" in "Section 5. Data Areas") ①

10	11-12	n - 1 to n + 1
0	GN number	Dictionary for Q pointer routines ②a

1 byte	2 bytes
Count of all GNs for Q-Routines associated with this file	Number of GN for string of Q-Routine GNs

②a Bytes 10-12 are present only if the Q-bit is on.

CD-NAME REFERENCE

n - 11 to n - 10	n - 9 to n - 8
PN number for STANDARD ERROR declarative ②	PN number for file header label ②

0	1	2 through n - 2	n - 1 to n + 1
25	n	Dictionary attributes of CD-name (see "CD ENTRY" in "Section 5.	dictionary pointer

VSAM FILE-NAME REFERENCE

0	1	2 through 9	10
26	n	Dictionary attributes of file (see "FD ENTRY" in "Section 5. Data Areas")	Count of all GNs for Q-routines associated with this file

11-12	13-14
GN number for string of Q-routine GNs	GN number for STANDARD ERROR declarative

15-22	23-24
Reserved	Pointer to dictionary entry for file

n - 1 to n + 1
Pointer to dictionary entry for item

DATA-NAME REFERENCE IN KEY CLAUSE (VSAM FILES)

0	1	2 to n - 6	n - 5
30	n	Dictionary attributes (see "LD ENTRY" in "Section 5. Data Areas")	Count of all GNs for Q-routines under item

n - 4 to n - 3	n - 2	n - 1 to n + 1
First GN number in series of all GN numbers for Q-routines under item	Index number	Pointer to dictionary entry for item

DATA-NAME REFERENCE

0	1	2 to n - 5 or n - 2
30	n	Dictionary attributes (see "LD ENTRY" in "Section 5. Data Areas")

n - 4
Count of all GNs for Q-Routines under item

n - 3 to n - 2
First GN number in series of all GN numbers for Q-Routines under item

NUMERIC LITERAL

0	1	2	3
32	n	Positions to left of decimal	Positions to right of decimal

4 through 1 + n
Literal in packed decimal format

FLOATING-POINT LITERAL

0	1	2-9
33	08	Literal represented as double-precision floating-point number

ALPHANUMERIC LITERAL

0	1	2 through 1 + c
34	c	Literal in EBCDIC

"EXHIBIT NAMED" NAME

0	1	2 through 1 + c
35	c	EBCDIC form of name used in EXHIBIT NAMED statement

INDEX-NAME REFERENCE

0	1	2	3-4	5-6
36	c	5	Index-name number	Length of subject

7-9
Pointer to dictionary entry for index-name

LISTING A-TEXT FOR PROCEDURE-NAMES

0	1	2 through 1 + c
37	c	EBCDIC procedure name; the preceding field is set to 1

LISTING A-TEXT FOR VERBS

0	1	2 through n	n + 1
37	n	EBCDIC verb	Alphabetic verb sequence number

DATENAME REFERENCE FOR OBJECT OF GIVING OPTION OF USE ERROR DECLARATIVE

0	1	through n + 1
38		Same as "Data-name Reference" (30) element above

"ALL" LITERAL LONGER THAN ONE CHARACTER

0	1	2 through c + 1
39	c	Alphanumeric value following ALL

CRITICAL PROGRAM BREAK

0	1
42	Break code (6)

VERB

0	1
44	Verb code (see note (3) under P0-text)

RELATIONAL CODE

0	1
50	06 (hex) = Equal
	08 = Greater than
	0A = Less than

PARENTHESIS

0	1
52	00 (hex) = Left parenthesis
	01 = Right parenthesis

ARITHMETIC OPERATOR

0	1
53	Operator code (7)

GENERATED PROCEDURE-NAME DEFINITION

0	1-2
88	GN number -- identifying number assigned to compiler-generated procedure-names from COMMON field GNCTR

COBOL WORD

0	1
54	Word code (see note (7) under P0-text)

GENERATED PROCEDURE-NAME REFERENCE

0	1-2
AA	GN number -- identifying number assigned to compiler-generated procedure-names from COMMON field GNCTR

SPECIAL NAMES

0	1
55	Code (5)

ERROR SYMBOL

0	1
B9	(8)

NFILES

0	1
56	Number of files in USING

CARD NUMBER

0	1-3
C1	Source card number (9)

COBOL WORD 2

0	2
57	Word code (see note (9) under P0-text)

- ① Dictionary attributes without count and major code fields.
- ② These fields appear only if the file-name reference is an operand in an OPEN statement. When a GN is not generated, the field contains zeros.
- ③ Pointer contents:
 

Bits	Contents
0-1	Unused
2-14	Dictionary section number
15-23	Displacement in section
- ④ Dictionary attributes with two fields removed. Bits 1-4 of the flag byte field overlay bits 1-4 of level number. Bits 5-8 overlay count field preceding major code field. In the level number field, if bit 5 is on, the level is 01; if bit 6 is on, the level is 77. Otherwise, the bits are off.

FIGURATIVE CONSTANT

0	1
75	EBCDIC value of figurative constant

STANDARD DATA-NAME REFERENCE

0	1
79	01 (hex) = LINE-COUNTER
	02 = PAGE-COUNTER
	05 = TALLY

In addition, the subfield of zeros starting at the tenth bit of the characteristics field is deleted for alphanumeric items and elementary items with either report or sterling report pictures.

In the case of data-name references to special registers, the addressing parameters field of the dictionary attributes contains an ID number according to the following schedule:

ID	Special Register
FFC000-FF0007	UPSI
FF0008	CURRENT-DATE
FF0009	TIME-OF-DAY
FF000B	SORT-RETURN
FF000C	SORT-CORE-SIZE
FF000D	SORT-FILE-SIZE
FF000E	SORT-MODE-SIZE
FF000F	LABEL-RETURN
FF0010	RETURN-CODE
FF0011	SORT-MESSAGE
FF0012	DAY
FF0013	TIME
FF0014	DATE
FF0015	WHEN-COMPILED

0C	procedures End of Report Writer procedures
0D	End of segment
0E	Communication Section
0F	Date-Compiled entry
F0	Security entry
F1	Identification Division
F2	Program-ID entry
F3	Author entry
F4	Environment Division
F5	Configuration Section
F6	Source-Computer entry
F7	Object-Computer entry
F8	Input-Output Section
F9	File-Control entry
FA	I-O-Control entry
FB	Special-Names Section
FC	Date-Written entry
FD	Installation entry
FE	Remarks entry

⑤

Bits	Contents
0	If 1, subject has variable length; bytes 5-6 contain VLC number.
1-3	Unused
4-7	1111

⑥

Code (hex)	Meaning
01	Data Division
02	File Section
03	Working-Storage Section
04	Linkage Section
05	Report Section
06	Procedure Division
07	Start of declaratives
08	End of declaratives
09	Start of debug packets
0A	Start of Q-Routines
0B	Start of Report Writer

⑦

Code (hex)	Operator
00	Addition
01	Subtraction
02	Multiplication
03	Division
04	Exponentiation
07	Unary minus

⑧

Error Symbol	When Used
COBOL word number	Reserved word used invalidly.
00 (hex)	Undefined or multiply-defined symbol found.

⑨ Bit 0 in byte 1 is on for a verb.

PROCEDURE P1A-TEXT

P1-A text is identical to P1-text with the following exceptions:

- ② Additional verb codes:  
 83 DEBUG transfer of control  
 8A USE FOR DEBUGGING debug verb  
 96 Debug subscript verb

① Procedure-name Reference

0	1	2-3
	Priority number	PN number
D0	of segment in	identifying
	which PN is	sequential number
	located	of source pro-
		cedure name
		assigned from
		PNCTR in COMMON



CEDURE IC-TEXT (P2 FORMAT)

e: The code byte of Procedure IC-text (P2 Format) is used to determine the length of the item:  
 For codes less than 40, the length is in the next byte.  
 For codes 40 through 7F, the length is two bytes.  
 For codes 80 through BF, the length is three bytes.  
 For codes C0 through FA, the length is four bytes except for code F9 whose length is three bytes.

E-NAME REFERENCE

0	1	Variable (9 to 16 bytes)	1 byte	2 bytes	n - 11 to n - 10
2	1	Dictionary attributes of file (see "FD ENTRY" in "Section 5. Data Areas") (1)	Count of all GNS for Q-Routines associated with this file	Number of GN for string of Q-Routine GNS	PN number for STANDARD ERROR declarative (2)

n - 9 to n - 8	n - 7 to n - 6	n - 5 to n - 4	n - 3 to n - 2	n - 1 to n + 1
PN number for file header label (2)	PN number for file trailer label (2)	PN number for end-of-volume label (2)	PN number for beginning-of-volume label (2)	Pointer to dictionary entry for file (3)

RB INFORMATION

0	1	2 through 1 + c
24	c	Follows verb string for EXAMINE, TRANSFORM, EVAL, ADD, SUBTRACT, MULTIPLY, DIVIDE, USE, UNSTRING, and DEBUG.

RB INFORMATION (VSAM)

0	1	2	3	4	5-7
24	c	0 ACB (3e)	Execution-time information (3f)	Compile-time information (3b)	Dictionary pointer to RECORD KEY dataname (3d)

NAME REFERENCE

0	1	2 through n - 2	n-1 to n+1
25	c	Dictionary attributes of CD-name (see "CD ENTRY" in "Section 5. Data Areas")	Dictionary pointer

VSAM FILE-NAME REFERENCE

0	1	2 through 9	10
26	n	Dictionary attributes of file (see "FD ENTRY" in "Section 5. Data Areas")	Count of all GNs for Q-routines associated with this file

11-12	13-14	15-22	23-24
GN number for string of Q-routine GNs	GN number for STANDARD ERROR declarative	Reserved	Pointer to dictionary entry for file

DATA-NAME INFORMATION FOR UNSTRING

0	1	2	3-5	6-9	10-11	12	13
2A	n	Type flag	Length	Base code	Displacement	Number of digits to right of decimal point	Scaling factor

Note: If bits 4-7 of the Type flag indicate an edited data item, the following fields are also generated.

0	1	2-4	5	6	7 - n + 1
2A	n	Size of data item	BLANK WHEN ZERO indicator	Number of bytes in edit mask	Edit mask (PICTURE clause for data item)

DATA-NAME REFERENCE

0	1	2 to n - 5 or n - 2	n - 4	n - 3 to n - 2	n - 1 to n + 1
30	n	Dictionary attributes of data-name (see "LD ENTRY" in "Section 5. Data Areas")	Count of all GNs for Q-Routines under item	First GN number in series of all GN numbers for Q-Routines under item	Pointer to dictionary entry for item

DATA-NAME REFERENCE FOR KEY CLAUSE

0	1	2 to n - 6 or n - 3	n - 4	n - 3 to n - 2	n - 2	n - 1 to n + 1
30	n	Dictionary attributes of data-name (see "LD ENTRY" in "Section 5. Data Areas")	Count of all GNs for Q-routines under item	First GN number in series of all GN numbers for Q-routines under item	Index ACB number	Pointer to dictionary entry for item

SUBSCRIPTED DATA-NAME REFERENCE

0	1	2 through n - 2	n - 1 to n + 1
31	n	Dictionary attributes of subscripted data-name (see "LD ENTRY" in Appendix D)	Pointer to dictionary entry

NUMERIC LITERAL (DECIMAL)

0	1	2	3	4 through 1 + n
32	n	Positions to the left of decimal	Positions to the right of decimal	Literal in packed decimal format

FLOATING-POINT LITERAL

0	1	2-9
33	08	Literal represented as double-precision floating-point number

ALPHANUMERIC LITERAL

0	1	2 through 1 + c
34	c	Literal in EBCDIC

"EXHIBIT NAMED" NAME

0	1	2 through 1 + c
35	c	EBCDIC form of name used in EXHIBIT NAMED statement

INDEX-NAME REFERENCE

0	1	2	3-4	5-6	7-9	10-11
36	n	(14)	Index-name number	Length of subject (14)	Pointer to dictionary entry for item (3)	Value in binary of integer specified in relative indexing clause (15)

LISTING A-TEXT FOR PROCEDURE-NAMES

0	1	2 through 1 + c
37	c	EBCDIC procedure name; the preceding bit is set to 1

LISTING A-TEXT FOR VERBS

0	1	2 through n	n + 1
37	n	EBCDIC verb	Alphabetic verb sequence number

MULTIPLE GN REFERENCE

0	1	2-3	4-5
38	04	GN number	GN number

FIGURATIVE CONSTANT "ALL" (Greater than 1 character)

0	1	Variable
39	c	Alphanumeric literal following ALL

SPECIAL NAMES

0	1	
55		Code (See Note 5) under P0-text

CRITICAL PROGRAM BREAK

0	1	
42		Break code 16

COBOL WORD 2

0	1	
57		Word code (See note 9) under P0-text

PHASE 4 OPTIMIZATION INFORMATION

0	1	
43		Type code 17

NFILES

0	1	
56		Number of files in USING

RELATIONAL CODE

0	1	
50	06 (hex)	= Equal
	08	= Greater than
	0A	= Less than
	0C	= Not less than

FIGURATIVE CONSTANT

0	1	
75		EBCDIC value of figurative constant

COBOL WORD

0	1	
54		Word code (See note 7) under PC-text

STANDARD NAME REFERENCE

0	1	
79	01 (hex)	= LINE-COUNTER
	02	= PAGE-COUNTER
	05	= TALLY

VERB

0	1	2
84		Verb code (see note 3) under PC-text) Count of elements that follow for this statement

GENERATED PROCEDURE-NAME DEFINITION

0	1-2	
88		GN number -- identifying number assigned to compiler-generated procedure-names from COMMON field GNCTR

GENERATED PROCEDURE-NAME REFERENCE

0	1-2	
AA		GN number -- identifying number assigned to compiler-generated procedure-names from COMMON field GNCTR

INTERMEDIATE RESULT REFERENCE

0	1-2
BA	IR number -- identifying number assigned to intermediate result

NUMERIC LITERAL (BINARY)

0	1-2
BB	Literal in binary format

TEMPORARY RESULT REFERENCE

0	1-2
BC	TR number

CARD NUMBER

0	1-3
C1	Source card number (18)

CARD NUMBER AS OPERAND OF FLOW VERB

0	1-3
C2	Source card number

PROCEDURE-NAME DEFINITION

0	1	2-3
C7	Priority number	PN number -- identifying sequential number of source procedure-name, assigned from COMMON field PNCTR

FILE-NAME REFERENCE

0	1-3
C8	Dictionary pointer

VARIABLE PROCEDURE-NAME DEFINITION

0	1	2-3
C9	Priority number of segment in which VN is located	VN number -- identifying number assigned to compiler-generated variable procedure-names from COMMON field VNCTR

PROCEDURE-NAME REFERENCE

0	1	2-3
D0	Priority number of segment in which PN is located	PN number -- identifying sequential number of source procedure-name, assigned from COMMON field PNCTR

PROCEDURE-NAME REFERENCE FOR XREF

0	1	2-3
D4	Priority number of segment in which PN is located	PN number -- identifying sequential number of source procedure-name, assigned from COMMON field PNCTR.

VARIABLE PROCEDURE-NAME REFERENCE

0	1	2-3
DB	Priority number of segment in which VN is located	VN number -- identifying number assigned to compiler-generated variable procedure-names from COMMON field VNCTR

GLOBAL TABLE REFERENCE (TYPE 1)

0	1	2
F9	Cell code for Task Global Table	Displacement in bytes from start of cell

GLOBAL TABLE REFERENCE (TYPE 2)

0	1	2-3
FA	Cell code for Task or Program Global Table	

① During phase 4 OPEN and CLOSE verb analysis, byte 2 (count and major code field) of the file-name reference element is changed to one of the following:

Bits	Contents	Meaning, if OPEN	Meaning, if CLOSE
0-3	0011	LEAVE	NO REWIND
	0001	REREAD	Default (REWIND)
	0000	DISP	Unused
	1000		REEL
4-7	0000	OPEN INPUT or CLOSED	
	1111	OUTPUT	
	0001	INPUT REVERSED	

② These fields appear only if the file-name reference is an operand in an OPEN or SORT statement. When a GN is not generated, the field contains zeros

③ Pointer contents:

Bits	Contents
0-1	Unused
2-14	Dictionary section number
15-23	Displacement in section

③a For VSAM READ, C = 6

Bit	Meaning
0	0 = SEQUENTIAL ACCESS or READ NEXT with DYNAMIC ACCESS
1-7	1 = RANDOM or DYNAMIC ACCESS
1-7	Unused

Bit	Meaning
0	0 = No duplicate string follows
1	1 = Duplicate string follows (READ INTO with MOVE only)
1-7	Unused

③d This field is used only for a READ verb with a KEY clause

- ③e Count field = 6 for READ verb, otherwise = 3
- ③f For START with KEY dataname clause, field = ACB#, otherwise = 0

Bits	Code	Meaning
0-1	00	Delimiter field
	01	Receiving field
	10	Delimiter-in field
	11	Count-in field
2-3	00	Unused
4-7	0000	Variable-length group
	0001	Alphanumeric
	0010	Alphanumeric, right-justified
	0011	Alphanumeric edited
	0100	Alphanumeric edited, right-justified
	0101	Numeric edited
	0110	External decimal, unsigned
	0111	External decimal with trailing overpunch
	1000	External decimal with leading overpunch
	1001	External decimal with trailing separate sign
	1010	External decimal with leading separate sign
	1011	Binary
	1100	Internal decimal, unsigned
	1101	Internal decimal, signed

- ⑤ If bits 4-7 of the Type flag are set to 0000, this field contains the VLC number. For all other codes, the field contains the SIZE, or length of the complete data item to be moved.

Byte	Contents
1	Base code
	X'0C' - BL
	X'14' - SBL
	X'28' - BLL
2	X'01'
3-4	BL number

This field is set to zeros for a subscripted data item.

- ⑦ This field is set to zeros for a subscripted data item. Otherwise it contains the displacement of the data item from the BL.
- ⑧ This field contains the number of actual digits to the right of a real or assumed decimal point. It is present only for a receiving or delimiter-in field. If this field is present, the scaling factor is set to zero.

- ⑨ The scaling factor represents the number of Ps to the left of the decimal point in a scaled integer. This field is present only for a receiving or delimiter-in field.

- ⑩ This field contains the total size of the data item, including all editing characters and decimal positions.
- ⑪ This field is set to X'FF' when the data item contained a BLANK WHEN ZERO clause; otherwise, it is set to X'00'.
- ⑫ Dictionary attributes with two fields removed. Bits 1-4 of the flag byte field overlay bits 1-4 of level number. Bits 5-8 overlay the count field preceding major code field. In the level number field, if bit 5 is on, the level is 01; if bit 6 is on, the level is 77. Otherwise, the bits are off.

In addition, the subfield of zeros starting at the tenth bit of the characteristics field is deleted for alphanumeric edited items and elementary items with report or sterling report pictures.

In the case of data-name references to special registers, the addressing parameters field of the dictionary attributes contains an ID number according to the following schedule:

ID	Special Register
FF0000-FFC007	UPSI-0 through UPSI-7
FF0008	CURRENT-DATE
FF0009	TIME-OF-DAY
FF000B	SORT-RETURN
FF000C	SORT-CORE-SIZE
FF000D	SORT-FILE-SIZE
FF000E	SORT-MODE-SIZE
FF000F	LABEL-RETURN
FF0010	RETURN-CODE
FF0011	DATE
FF0012	DAY
FF0013	TIME
FF0015	WHEN-COMPILED

If this data-name reference contains a subscript or index address calculation ID number, the high-order bit is on, bits 1-7 contain zeros, and bits 8-23 contain the ID number. The high-order bit is turned on by phase 4 when it assigns the ID number.

- ⑬ Addressing parameters field in attributes has been replaced by unique subscript identifier element to match entry in DEFSBS table.
- ⑭ Contents of byte 2:

Bit	Contents
0	0 = Bytes 5-6 contain fixed length of subject.
1	1 = Subject has variable length; bytes 5-6 contain VLC number.

15 For a relative indexing clause, phase 4 adds this field to the element. Phase 50 recognizes the presence of this field from the number of bytes given in byte 1 of the element.

Code (hex)	Meaning
01	Data Division
02	File Section
03	Working-Storage Section
04	Linkage Section
05	Report Section
06	Procedure Division
07	Start of declaratives
08	End of declaratives
09	Start of debug packets
0A	Start of Q-Routines
0B	Start of Report Writer procedures
0C	End of Report Writer procedures
0D	End of segment
0E	Communication Section
0F	Date-Compiled entry
F0	Security entry
F1	Identification Division
F2	Program-ID entry
F3	Author entry
F4	Environment Division
F5	Configuration Section
F6	Source-Computer entry
F7	Object-Computer entry

Code	Meaning
F8	Input-Output Section
F9	File-Control entry
FA	I-O-Control entry
FB	Special-Names Section
FC	Date-Written entry
FD	Installation entry
FE	Remarks entry

Code (hex)	Meaning
02	Precedes a Procedure-name definition element equated to a VN for an ALTER verb
04	Precedes a Variable procedure-name reference element at a PERFORM verb exit
05	Precedes a Generated procedure-name definition element at the returning point of a performed procedure

18 Bit 0 in byte 1 is on for a verb.

19 Used to write an XREF element for procedure-name B in the following cases:

- PERFORM A THRU B.
- ALTER A TO PROCEED TO B.
- A. GO TO B. (where A is altered)

Code (hex)	Meaning
02	SAVE AREA
04	SWITCH
06	Unused
08	DEBUG

Code (hex)	Meaning
00	DCBADR
04	VLC
08	ONCTL
0C	PFMCTL
10	PFMSAV
14	DECBADR
18	XSA
1C	PARAM
1D	Single-precision floating-point
1E	Double-precision floating-point
20	WORKING CELLS
24	Temporary storage
28	XSASW
2C	BL, SBL, BLL
30	VIRTUAL
34	FIB

22 Except when the code for the preceding field 11 is 2C, 1D, or 1E, this field contains the identifying number from one of the COMMON counters as described in "Section 5. Data Areas." When the preceding field is 2C, this field contains the i and k fields of the addressing parameters, as follows:

Bits	Field	Value	Meaning
0-3	i	0000	BL
		0001	BLL
		0100	SBL
4-7	-	0000	Unused
8-15	k		Base locator number assigned from corresponding COMMON counter.

When the code for the preceding field is 1D or 1E, this field contains zeros.



ATM-TEXT

ATM-text is a subset of Procedure IC-text (P2 format). It may contain all elements of P2-text except the following:

Code Element Name

21 FILE-NAME REFERENCE  
 24 VERB INFORMATION  
 33 FLOATING-POINT LITERAL  
 35 "EXHIBIT NAMED" NAME

BA INTERMEDIATE RESULT REFERENCE  
 BC TEMPORARY RESULT REFERENCE  
 C9 VARIABLE PROCEDURE-NAME DEFINITION  
 D0 PROCEDURE-NAME REFERENCE  
 D4 PROCEDURE-NAME REFERENCE FOR XREF  
 DB VARIABLE PROCEDURE-NAME REFERENCE  
 F9 GLOBAL TABLE REFERENCE (TYPE 1)

For the formats of ATM-text elements, see the formats under "Procedure IC-text (P2 Format)."

PROCEDURE A-TEXT

PN AND GN DEFS AND PROGRAM BREAKS

0	1	2
27   n   24, C7, or 88 elements		

MISCELLANEOUS A-TEXT

0	1	2
28   n   All elements except 24, C7, 88     (see 27) and 00 (see 29)		

E-TEXT

0	1	2
29   n   00 elements		

CARD NUMBER

0	1-3
2C	Source card number (1)

SOURCE PROCEDURE-NAME DEFINITION

0	1	2-3
30	(2)	PN number -- number assigned from PNCTR in COMMON

GENERATED PROCEDURE-NAME DEFINITION

0	1-2	
34	(2)	GN number -- number assigned from GNCTR in COMMON

VARIABLE PROCEDURE-NAME DEFINITION

0	1	2-3
38	(2)	VN number -- number assigned from VNCTR in COMMON

EBCDIC PROCEDURE-NAME GENERATOR

0	Consists only of this one field.	
	Used to create in-line DC	
3C	instruction for current card number to be used by the TRACE verb.	

MACRO-TYPE INSTRUCTION

0	1	2
44	Type Code (3)	Priority number if type code is 4C

OPERATION CODE

0	1
48	Machine operation code. 00 (hex) used for CNOP

2
Value of second instruction byte: condition code, length, register, or immediate field (1a)

PROCEDURE-NAME REFERENCE

0	1	2-3
4C	(2)	PN number assigned at point of definition from PNCTR (COMMON)

GENERATED PROCEDURE-NAME REFERENCE

0	1-2	
50	(2)	GN number assigned at point of definition from GNCTR (COMMON)

VARIABLE PROCEDURE-NAME REFERENCE

0	1	2-3
54	(2)	VN number assigned at point of generation from VNCTR (COMMON)

VIRTUAL REFERENCE

0	1-2
58	VIR number assigned to source CALL statement operand from VIRCTR (COMMON)

ADDRESS REFERENCE

0	1-3	4-6
78	Addressing parameters (7)	Dictionary pointer (8)

BASE LOCATOR REFERENCE

0	1	2
5C	Type code (4)	BL or BLL number assigned from BLCTR or BLLCTR in COMMON

EBCDIC DATA-NAME REFERENCE

0	1	2 through 1 + c
7C	c	Data-name in EBCDIC (14)

GLOBAL TABLE STANDARD AREA REFERENCE

0	1	2
60	Type code (5)	Displacement in bytes from start of specified area

ADDRESS INCREMENT

0	1-3
80	Value that is to modify an address. Negative value in 2's complement.

GLOBAL TABLE VARIABLY-LOCATED AREA REFERENCE

0	1	2-3
64	Type code (6)	Identifying number of item within specified area

RELATIVE ADDRESS

0	1	2
84	Code for object module field (9)	Size, in bytes, of address constant

LITERAL REFERENCE

0	1-2
68	Number of literal

3-4
Number of item in specified field

DC DEFINITION

0	1	2 through c + 1
6C	c	Actual constant

SPECIAL PHASE 6 ELEMENTS

(10)

REGISTER SPECIFICATION

0	1
A0	Register number: 00 through CF (hex)

BASE AND DISPLACEMENT

0	1-2	
70	Bits	Contents
	0-3	Register number
	4-15	Displacement

INCREMENTED ADDRESS

0	1-3	4-6
A4	Addressing parameter (7)	Dictionary pointer (8)

7-9
Value of increment

SPECIAL PHASE 6 ELEMENTS

(10)

CALLING SEQUENCE DISPLACEMENT

0	1	2
B0	Code for object module field (11)	Base code switch (12)

3-4
Number of item in specified field

CALLING SEQUENCE DICTIONARY POINTER

0	1-3
B4	Pointer to dictionary entry for file-name, data-name, or subscripted data-name. (8)

FILE OR SUBSCRIPT REFERENCE ELEMENT

0	1-3
B8	Pointer to dictionary entry for file (8)

PARAMETER FOR CALL TO SEGMENTATION AND GO TO DEPENDING ON SUBROUTINES

0	1	2-3
BC	Priority	PN number

PHASE 50 OPTIMIZATION INFORMATION

0	1
C0	Type code (13)

RPT-ORIGIN

0	1-2
D4	GN number

(1) Bit 0 in byte 1 is on for a verb.

(1a)

Code (hex)	Assembler equivalent
04	CNOP 0,4
08	CNOP 2,4
0c	CNOP 0,8
10	CNOP 2,8
14	CNOP 4,8
18	CNOP 6,8

(2) Byte 1 contains priority number of segment within which the procedure-name is located.

③ Code (hex)	<u>Meaning</u>
00	EQUATE -- equates variable procedure-names to initial value; followed by procedure-name reference.
04	ENTRY -- defines entry point; followed by EBCDIC data-name reference giving entry point name.
08	BLCHNG -- specific contents of base locator changed; followed by base locator reference.
0C	ENDOPT -- indicates end of register optimization.
10	DECLARATIVES START -- indicates beginning address of Declaratives Section; produced only if the SYMDMP or STATE option is in effect.
18	ADCON -- defines address constant; followed by relative address reference, procedure-name reference, calling sequence displacement element, or calling sequence dictionary pointer to which adcon points.
20	START -- identifies first executable instruction.
24	DC -- identifies constant; followed by DC definition element.
28	RESERVE -- specifies registers not to be used by phase 6 or by phases 62, 63, and 64; followed by register specification element.
2C	FREE -- indicates registers no longer reserved; followed by register specification element.
30	DESTROY -- indicates that contents of register were destroyed; followed by register specification element.
34	INIT -- indicates when permanent registers must be loaded.
38	ORIGIN -- indicates where to set location counter for overlaying USE BEFORE REPORTING code; followed by generated procedure-name reference.
3C	REORIGIN -- indicates that the reset location counter is to be reset.
40	Q-BEGIN -- indicates start of Q-Routine coding; destroys permanent register assignment.
44	Segmentation control break.
4C	Segment initialization (1-byte priority number in next field).
50	Dummy procedure-name reference element to force an XREF element to be written for the

following procedure-name reference element.

④ Bits	<u>Code</u>	<u>Meaning</u>
0-3	0000	Unused
4-7	0000	BL
	0001	BLL

⑤ Code (hex)	<u>Meaning</u>
02	SAVE-AREA
04	SWITCH
06	TALLY
08	SORT-SAVE
0A	USDBGINF

⑥ Code (hex)	<u>Meaning</u>
00	DCBADR
04	VLC
08	ONCTL
0C	PFMCTL
10	PFMSAV
14	DECBADR
18	XSA
1C	PARAM
20	WORKING CELLS
24	TS
28	XSASW
2C	SUBADR
30	TS-2
34	FIB
3C	SAVE AREA-2
40	SAVE AREA-3
44	RPTSAV AREA
48	OVERFLOW
50	CHECKPT CTR
54	TS-3
58	TS-4
5C	IND
60	DBGTRA
64	DBGCRD

⑦ Bits	<u>Field</u>	<u>Code</u>	<u>Content or Meaning</u>
0-3	i	0000	BL
		0001	BLL
		0011	Address of data-name is in register specified in bits 12-15. Referred to by zero displacement from register.
		0100	SBL
		0110	subscript cell
4-15	d		Displacement from start of area controlled by base locator. If a register is specified in bits 0-3, however, bits 4-11 contain zeros, and bits 12-15 contain the register number.
16-23	k		Base locator number assigned from corresponding COMMON counter.

⑧	Bits	Contents
	0-1	Unused
	2-14	Dictionary section number
	15-23	Displacement in section

⑨	Code (hex)	Meaning
	00	INIT1
	04	TALLY
	08	PARAM
	0C	BL
	14	SBL
	1C	VLC
	28	BLL
	2C	LITERAL
	30	INIT3
	34	CHECKPT CTR
	38	PGT
	3C	TGT
	40	INIT2
	44	START (first executable instruction identified by START)
	48	PN
	4C	VIRTUAL
	50	GN
	58	VN
	5C	VNI
	60	SUBADR
	64	Temporary Storage
	68	External adcon (for address of transient area in segmented program)

⑩ The following special phase 6 A-text elements are generated and then used by phase 6 to generate MVC instructions to initialize VN cells in the TGT:

Identifier	Byte	Text Element
	90	A 4-byte element that references the VNI cell in the PGT.
A8		A 1-byte element that indicates that the value in the P6PLUS field should be used as the "plus" element of the MVC instruction.
AC		A 1-byte element that indicates that the value in the P6LNG field should be used as the "length" element of the MVC instruction.

⑪	Code (hex)	Meaning
	00	INIT1
	04	TALLY
	08	PARAM
	0C	BL
	10	SAVE2
	14	SBL
	18	FIB
	1C	VLC
	20	PN
	24	GN
	28	BLL
	2C	LITERAL
	30	DCB
	60	SUBADR
	64	Temporary storage

⑫	Code (hex)	Meaning
	00	No preceding base code.
	01	Base code precedes this element.

⑬	Code (hex)	Meaning	Phase Produced by
	01	Precedes a load instruction which is not followed by a branch instruction	50,51
	02	Precedes any possible entry point for which addressability must be established	4,51
	03	Precedes a load instruction for a PN or GN which must be processed with an address constant cell in the PGT	50,51
	04	Precedes a Variable procedure-name reference element at a PERFORM verb exit	4
	05	Precedes a Generated procedure-name definition element at the returning point of any performed procedure except a PERFORM...TIMES procedure in a nonsegmented program	4,51
	06	Precedes a load instruction for a PN or GN which is followed by an unconditional branch instruction	50,51

⑭ Must be preceded by either a 4404 or a 2C element.

OPTIMIZATION A-TEXT

VIRTUAL DEFINITION

0	1	2-3
00	00	VIR number assigned from VIRCTR in COMMON

4-11
External-name in operand of CALL statement, left-justified, padded with blanks.

LITERAL DEFINITION

0	1	2	3 through 2 + c
04	Type	c	Value of literal code
	①		

GENERATED PROCEDURE-NAME EQUATE STRING (NON-OPTIMIZER VERSION)

0	1	2-3
08	n/2	First GN number assigned (Number of fields to follow)

[ Variable number of 2-byte fields containing GN numbers ]
--

n through n + 1
Last GN number assigned to same location as others in string

SOURCE PROCEDURE-NAME EQUATE STRING (NON-OPTIMIZER VERSION)

0	1	2-3
0C	n/2	First PN number assigned (Number of fields to follow)

[ Variable number of 2-byte fields containing GN numbers ]
--

n through n + 1
Last GN number assigned to same location as others in string

DISPLAY LITERAL DEFINITION

0	1	2	3 through 2 + c
10	Type	c	Value of literal code
	①		

LITERAL DEFINITION (Larger than 255)

0	1	2-3	4-3 + c
54	Type	C	Values of literal Code
	①		

SEGMENTATION ELEMENT

0	1	2-3
14	Priority number of segment to which VN belongs	VN number

GNUREF ELEMENT

0	1-2
1C	GN number for AT END or INVALID KEY branches, or GNS at REPORT-ORIGIN

PNUREF ELEMENT

0	1-2
20	PN number for PNs following TO PROCEED TO in ALTER statements or Declaratives PN number

PROCEDURE-NAME REFERENCE

0	1	2-3
4C	2	PN number assigned at point of definition from PNCTR (COMMON)

GN-VN ELEMENT PERFORM VERB

0	1-2	3-4
24	GN number -- associated with return point of a performed procedure	VN number -- associated with PERFORM exit

GENERATED PROCEDURE-NAME REFERENCE

0	1-2
50	GN number at point of definition from GNCTR (COMMON)

VARIABLE PROCEDURE-NAME EQUATE  
PROCEDURE-NAME OR VARIABLE PROCEDURE-NAME  
EQUATE GENERATED PROCEDURE-NAME ELEMENT  
(OPTIMIZER VERSION)

VARIABLE PROCEDURE-NAME DEFINITION

0	1	2-3
38	2	VN number -- number assigned from VNCTR in COMMON

MACRO-TYPE INSTRUCTION

0	1
44	00
	3

1	Bits	Code	Meaning
	0-1	00	No boundary requirement
		01	Halfword boundary
		10	Fullword boundary
		11	Doubleword boundary
	2	1	Floating-point number
	3	1	EBCDIC numeric value
	4	1	Binary number
	5	1	Packed decimal number
	6	1	EBCDIC character string
	7	1	Hexadecimal number

2) Byte 1 contains the priority number of the segment within which the procedure-name is located.

3) The code 00 indicates EQUATE.



PROCEDURE A1-TEXT

The following elements of Procedure A1-text are identical to their counterparts in Procedure A-text.

Code	Element Name
2C	CARD NUMBER
30	SOURCE PROCEDURE-NAME DEFINITION
34	GENERATED PROCEDURE-NAME DEFINITION
38	VARIABLE PROCEDURE-NAME DEFINITION
3C	EBCDIC PROCEDURE-NAME GENERATOR
44	MACRO-TYPE INSTRUCTION (1)
48	OPERATION CODE
4C	PROCEDURE-NAME REFERENCE (2)
50	GENERATED PROCEDURE-NAME REFERENCE (2)
54	VARIABLE PROCEDURE-NAME REFERENCE
58	VIRTUAL REFERENCE
5C	BASE LOCATOR REFERENCE
60	GLOBAL TABLE STANDARD AREA REFERENCE
64	GLOBAL TABLE VARIABLY-LOCATED AREA REFERENCE
68	LITERAL REFERENCE
6C	DC DEFINITION
70	BASE AND DISPLACEMENT
7C	EBCDIC DATA-NAME REFERENCE
84	RELATIVE ADDRESS
A0	REGISTER SPECIFICATION
B0	CALLING SEQUENCE DISPLACEMENT
B4	CALLING SEQUENCE DICTIONARY POINTER
B8	FILE REFERENCE ELEMENT
BC	SEGMENTATION AND GO TO DEPENDING ON SUBROUTINE CALL PARAMETER

The SPECIAL PHASE 6 ELEMENTS are also present in Procedure A1-text and are processed by phase 64 if the OPT option is specified.

ADDRESS REFERENCE

0	1-3	4-6
78	Addressing parameters (3)	Dictionary pointer (4)

ADDRESS INCREMENT

0	1-3	4
80	Value that is to modify an address	Code (5)

BLOCK NUMBER

0	1
C4	Block number

PROCEDURE BASE REGISTER ELEMENT FOR PN'S

0	1-2
C8	PN number

PROCEDURE BASE REGISTER ELEMENT FOR GN'S

0	1-2
CC	GN number

BASE DISPLACEMENT DATA-NAME

0	1-2	3-5
D0	Addressing parameters (6)	Dictionary pointer (5)

① A MACRO-TYPE INSTRUCTION element with a 44 byte code (segmentation control break) has an added byte which contains the priority number.

② Present only for those PNs and GNs that have address constant cells in the PGT.

Bits	Contents
0-1	Unused
2-14	Dictionary section number
15-23	Displacement in section

Code	Meaning
00	LA instruction not to be generated
0E	LA instruction to be generated, using register 14 as base register
0F	LA instruction is to be generated, using register 15 as base register.

Bits	Field	Code	Content or Meaning
0	i	0	Use register 14 to address item. (A load instruction is generated if bits 2 and 3 are on.)
		1	Use register 15 to address item. (A load instruction is generated if bits 2 and 3 are on.)
1-3		000	BL
		001	BLL
		011	Address of data-name is in register specified in bits 12-15. Referred to by zero displacement from register.
		100	SBL
		110	Subscript cell
4-15	d		Displacement from start of area controlled by base locator. If a register is specified in bits 0-3, however, bits 4-11 contain zeros, and bits 12-15 contain the register number.
16-23	k	1	BL or BLL number assigned from BLCTR, or BLCTR in COMMON
		2	Subscript cell number
		3	SBL number assigned from SBLCTR in COMMON

⑥ For data-names with permanently addressable BLs, phase 63 changes the

Address reference element to a Base displacement data-name element, using the permanently assigned register number from the BLASGTBL table and the displacement given in the Address reference element. The contents of the field are:

Bits	Contents
0-3	Base register
4-15	Displacement from start of area controlled by base locator

LISTING A-TEXT

LISTING A-TEXT FOR PROCEDURE-NAMES

0	1	2 through 1 + c
7C	c	EBCDIC procedure-name;
		bit 0 of the preceding field
		is set to 1

LISTING A-TEXT FOR VERBS

0	1	2 through n	n + 1
7C	n	EBCDIC verb	Alphabetic verb
			sequence number

END OF LISTING A-TEXT

0
01

E-TEXT

MESSAGE DEFINITION

0	1	2	3-4
00	07	00	Identifying
			message number

5-7	8
Card number	Bits Contents
	0-3 Severity code
	4-7 Phase number

MESSAGE PARAMETER

0	1	2
00	n	IC-text identifier for parameter ①

3 through n + 1

Either the actual value to be inserted, or a pointer to the PARTBL field containing the value.

① Code (hex)	Meaning
05	Alphanumeric literal
22	Alphanumeric literal
23	EBCDIC name
34	Alphanumeric literal
42	Critical program break
43	Level number
44	Verb
50	Relational
52	Parenthesis
53	Arithmetic operator
54	COBOL word
57	COBOL word
75	Figurative constant
79	Standard data-name
87	Procedure-name definition
F9	Global Table reference, Type 1
FA	Global Table reference, Type 2
FE	Dictionary pointer (not an IC-text element)

XREF-TEXT

DEF-TEXT ELEMENT FOR DATA-NAME DEFINITION

0	1-3	4-6	7	8 to c + 7
48	Card number	Pointer to dictionary entry for data-name	c	External-name in EBCDIC

DEF-TEXT ELEMENT FOR PROCEDURE-NAME DEFINITION

0	1-2	3-5	6	7 to c + 6
4C	PN number	Card Number	c	External-name in EBCDIC

DEF-TEXT ELEMENT FOR VERB DEFINITION

0	1	2-3	4	5
48	00	Number of occurrences	E0	Alphabetic verb sequence number

6	7	8	9 to c + 7
00	c	FB	Verb text

END OF REF-TEXT and  
END OF E-TEXT

0
77
⑤

REF-TEXT FOR VERBS

0-2	3	4-5
Card Number ⑦	E0	Verb code ⑥

BEGINNING OF REF TEXT

0-1
F1F1
①

REF-TEXT

0-2	3-5
Pointer to dictionary entry for data-name or PN number	Card number ②

DEBUG-TEXT

CARDLOC

0	1-3	4-6
10	Card number	Contents of LOCCTR when card encountered

ENDSEG

0	1-3	4-6
20	Zeros	Contents of LOCCTR after processing last verb of segment

SEGMENT

0	1
30	Priority number
	(3)

DISCONTINUITY

0
40
(4)

- ① Written by phase 60 or 64; read by phase 6A.
- ② Bit    Meaning  
     0    0 = Reference is for data-name  
     1    1 = Reference is for procedure-name
- ③ For an unsegmented program, only one priority element is written; the priority number is set to zero.
- ④ Indicates beginning of a noncontiguous section within a segment.
- ⑤ End-of-REF-text on file 3 and end-of-E-text if spilled on file 3. Written by phase 60 or phase 64.
- ⑥ The alphabetic verb sequence number followed by X'00'.
- ⑦ The high-order bit is always set to 0.

DICTIONARY ENTRY FORMATS

Dictionary handling is performed by ACCESS routines described in "Appendix A: Table and Dictionary Handling." Phases 1B, 22, 21, 25, and 3 use the dictionary.

Phase 1B enters procedure-names and their attributes in the dictionary.

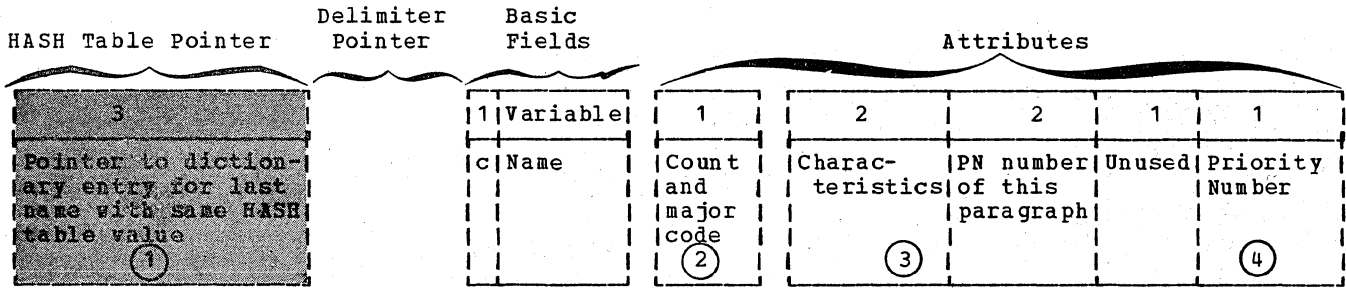
Phases 22 and 21 enter data-names and their attributes in the dictionary. If the SYMDMP option is in effect, phase 25 uses the dictionary to build the DATATAB and OBODOTAB tables for the Debug data set.

Phase 3 replaces data-names and procedure-names in Procedure Division statements with their dictionary attributes.

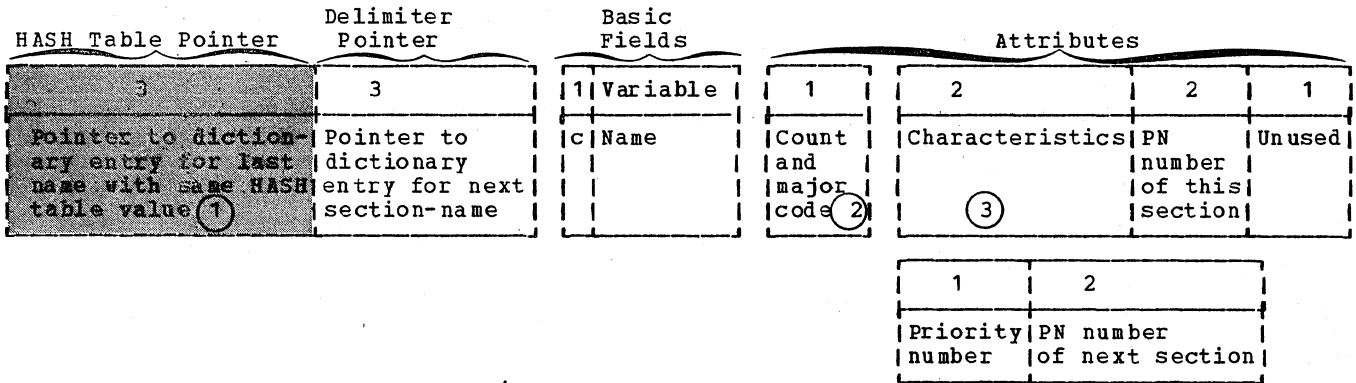
NOTES ON DICTIONARY ENTRY FORMATS

- The top row of figures shows the number of bytes in the field.
- ~~Boxes that are shaded~~ define optional fields or a series of similar fields.
- c = the number of bytes in the field that follows.
- n = the total number of bytes that follow in the remainder of the entry.
- 1b = this field is one byte long.

PROCEDURE-NAME (PARAGRAPH) ENTRY



PROCEDURE-NAME (SECTION) ENTRY



- (1) **Bits**    **Contents**
- 0-1    00 = Neither HASH table nor delimiter pointer
  - 01 = Delimiter pointer
  - 10 = HASH table pointer not followed by delimiter pointer
  - 11 = HASH table pointer followed by delimiter
  - 2-14    Dictionary section number
  - 15-23    Displacement in section

- (2) **Bits**    **Contents**
- 0-3    Number of bytes in attributes field. If 0, see variable information field under "LD ENTRY."
  - 4-7    **Code**        **Meaning**
  - 0000    LD entry under FD
  - 0001    LD entry under SD
  - 0010    LD entry under CD
  - 0011    VSAM file
  - 0100    LD entry in Working-Storage
  - 0101    LD entry in Linkage Section
  - 0110    LD entry in Report Section
  - 0111    LD entry under FD with APPLY WRITE-ONLY (after phase 3)
  - 1000    FD entry
  - 1001    SD entry
  - 1010    CD entry
  - 1011    FD entry with ASCII
  - 1100    Condition-name entry
  - 1101    Procedure-name entry
  - 1110    RD entry
  - 1111    Index-name entry

- (3) **Bit**    **Meaning, if on**
- 0    SORT/MERGE input/output procedure
  - 1    Section-name
  - 2    Referred to by PERFORM
  - 3    Referred to by ALTER
  - 4    Procedure-name of simple GO TO
  - 5    Procedure-name of EXIT
  - 6    Procedure-name following TO PROCEED TO in an ALTER statement
  - 7    Unused
  - 8    Referred to by DEBUG
  - 9    Defined in DEBUG
  - 10    Dummy section-name
  - 11    Defined in Declaratives Section or in DEBUG statement referred to such a section. Bits 12-1 describe type of section.
  - 12    Declarative error routine
  - 13    Declarative label routine
  - 14    Declarative for DEBUGGING
  - 15    Declarative report section

- (4) Not present in dictionary added by phase 30.

ENTRY

ASH Table Pointer	Delimiter Pointer	Basic Fields	Attributes				
3	3	1 Variable	1	1	1	1	
Pointer to Dictionary entry for last name with same HASH table value (1)	Pointer to next entry after last LD entry for this file (1)	c Name	Count and major code (2)	Flag byte (4)	First BL number for VSAM file	DCB number or FIB number (5)	Access method and I/O specifications (5a)

Non-VSAM file

1	2	1
DECB number	Maximum record length	Count and I/O options (6)

VSAM file

1	2	1
IND2TBL entry count	Displacement of first entry in IND2TBL	Number of BLs needed for this file

All files

3	1
idk addressing parameters for ACTUAL KEY (direct file only) (7)	Length of ACTUAL KEY minus four (direct file only)

3
idk addressing parameters for data-name in APPLY REORG-CRITERIA clause (BISAM file only) (7)

QSAM File

1
Option byte (7A)

④ Bits	Contents
0	1 = Q-Routine indication
1-2	00 = Format V 01 = Format F 10 = Format U 11 = Format S
3	USE AFTER ERROR
4-7	Number of BLS needed for this file

⑤ Bits	Access Method	Clause Specified
		Bit if bit is on
		4 BEFORE ADVANCING
		5 AFTER POSITIONING
0000	QSAM	6 APPLY WRITE-ONLY
		7 AFTER ADVANCING
		4 START
0001	QISAM	5-7 Unused
		4 APPLY REORG-CRITERIA
		5 TRACK AREA
0010	BISAM	IS... CHARACTERS
		6 APPLY CORE-INDEX
		7 FREE
		4 NOMINAL or ACTUAL KEY
0100	BSAM	5 Direct organization
		6-7 Unused
		4 File organization 'D'
1000	BDAM	5 Direct organization
		6 TRACK LIMIT
		7 Unused

⑤a VSAM entry

Bit	Contents
0	1 = ACCESS IS SEQUENTIAL
1	1 = ACCESS IS RANDOM
2	1 = ACCESS IS DYNAMIC
4	Reserved
5	Relative VSAM
6	1 = VSAM indexed file
7	1 = VSAM addressed sequential file

⑥ Bits	Contents
0-3	Number of bytes that follow
4	1 = SAME RECORD AREA
5	1 = SAME AREA
6	1 = User label
7	1 = nonstandard label

⑦ Location of item in data area of object module:

Bits	Field	Meaning
0-3	i	Type of BL containing base address of area: 0000 = BL 0001 = BLL 0100 = SBL
4-15	d	Displacement from base address
16-23	k	BL number

⑦A This extra byte is present for all QSAM files after count and input/output opt:

Bit	Meaning
0	LINAGE
2	FILE STATUS



ENTRY

ASH Table Pointer	Delimiter Pointer	Basic Fields	Attributes					
3	3	1 Variable	1	1	1	2	2	1
Pointer to dictionary entry for last name with same HASH table value (1)	Pointer to next entry after last LD entry for this file (1)	c Name	Count and major code (2)	Flag byte	First BL number	Maximum record length	Minimum record length	0 (9a)

ENTRY

ASH Table Pointer	Delimiter Pointer	Basic Fields	Attributes	
3	3	1 Variable	1	7
Pointer to dictionary entry for last name with same HASH table value (1)	Pointer to next entry after last LD entry for this file (1)	c Name	Count and major code (2)	0

Bits	Contents
0	1 = SAME RECORD AREA
1-2	00 = Format V or S 01 = Format F 10 = Format U
3	1 = ASCII collating sequence
4-7	Number of BLLs needed for this file.

LD ENTRY

HASH Table Pointer	Delimiter Pointer	Basic Fields	Attributes				
3	3	1 Variable	1	1	3	1	Variable
Pointer to Dictionary entry for last name with same HASH table value (1)	If group item, pointer to next entry after last LD entry for this group (1)	c Name	Count and major code (2) (13)	Minor code and flags (9)	Addressing parameters for item (10)	Flag byte (11)	Level number and variable information (12)

CD ENTRY

HASH Table Pointer	Delimiter Pointer	Basic Fields	Attributes				
	3	1 Variable	1	1	3	1	2
Pointer to Dictionary entry for last name with same HASH table value (1)	Pointer to next entry after last LD entry for this CD (1)	c Name	Count and major code (2)	CD number (14)	Addressing parameters for item (10)	Flag byte (15)	Number of occurrences of DESTINATION TABLE

(9) Bits	Code	Operand's Characteristics
0-3	0000	Error detected for this operand
	0001	Fixed-length group
	0010	Alphabetic
	0011	Alphanumeric
	0100	Variable-length group
	0101	Report item
	0110	Sterling report item
	0111	Usage is index
	1000	External decimal
	1001	External floating-point
	1010	Internal floating-point
	1011	Binary
	1100	Internal decimal
	1101	Sterling nonreport

1110	Alphanumeric edited
1111	Unused
4-5	Code Subscripts Required
00	None
01	1
10	2
11	3

6 1 = OCCURS clause in this item  
 7 1 = REDEFINES clause for this item

(9a) 0 = 1 SRA

(10) Location of item in data area of object module:

<u>Bits</u>	<u>Field</u>	<u>Meaning</u>
0-3	i	Type of BL containing base address of area: 0000 = BL 0001 = BLL 0100 = SBL
4-15	d	Displacement from base address
16-23	k	BL number

⑪

<u>Bits</u>	<u>Meaning, if on</u>
0	INDEXED BY
1	SYNCHRONIZED
2	Subject of key
3	If bit 2 = 1 0 = DESCENDING 1 = ASCENDING
4	Report WITH CODE
5	DEBUGGED item is subscripted
6-7	<u>Contents</u> 00 = SIGN is overpunch trailing 01 = SIGN is overpunch leading 10 = SIGN is separate trailing 11 = SIGN is separate leading

⑫

<u>Bits</u>	<u>Contents</u>
0-5	Level number
6	1 = Q-Routine is required
7-end	Variable, according to Table 18.

⑬ During phase 3 operations, the flag byte field is removed. The first four bits overlay the first four bits of the level number, and the last four bits overlay the count field preceding the major field. In the level number field, if bit 4 is on, the level is 01; if bit 5 is on, the level is 77. Otherwise, these bits are off.

⑭ Each CD is assigned an identifying number beginning with the last DCB number plus 1.

⑮

<u>Bits</u>	<u>Meaning</u>
0	0 = INPUT 1 = OUTPUT
1-5	Unused
6	1 = Object of OCCURS...DEPENDING ON under CD entry
7	Unused

Characteristic	Bits	Contents
Fixed-length Group Item	1-17	Length of group.
Elementary Alphabetic or Alphanumeric Item	1 2-17 18-32* 33-41*	1, if JUSTIFIED RIGHT. Length of item. Dictionary section number of entry for group in which this item is included, if the group contains an OCCURS clause. Displacement in section.
External Decimal or Internal Decimal or Binary	1 2-9 10-17 18-32* 33-41*	1, if PICTURE contains S. Signed number. If positive, the number is the total of Ps plus 9s to the right of decimal point. If negative, it is the number of Ps to the left of decimal. If 0, the item is an integer. Number of decimal digits. Dictionary section number of entry for group in which this item is included, if the group contains an OCCURS clause. Displacement in section.
Internal Floating-point	1-16 17 18-32* 33-41*	Unused 0 = Short form 1 = Long form Dictionary section number of entry for group in which this item is included, if the group contains an OCCURS clause. Displacement in section.
External Floating-point	1 2 3 4-9 10-17 18-32* 33-41*	0 = Mantissa blank when positive 1 = Plus sign when positive Same for exponent sign 0 = Implied decimal point 1 = Real decimal point Scale of mantissa. Total length. Dictionary section number of entry for group in which this item is included, if the group contains an OCCURS clause. Displacement in section.
Sterling Non-Report Elementary Item	1 2 3 4-9 10-14 15-17 18-25 or 18-32* 33-41* 42-49*	0 = BSI shillings 1 = IBM shillings 0 = 1-character pence 1 = 2-character pence 0 = BSI pence 1 = IBM pence Number of decimal places to the right of V in pence field. Number of pound field digits. 000 = No sign specified. 001 = Sign on high-order pound character 010 = Sign on low-order pound character 011 = Sign on high-order shilling character 100 = Sign on low-order pence character 101 = Sign on low-order decimal position in pence field (Values 110 and 111 are unused.) Number of character positions. or Dictionary section number of entry for group in which this item is included, if the group contains an OCCURS clause. Displacement in section. Number of character positions.
*Entry contains these fields only if item is in a group with an OCCURS clause. The fields are not present after phase 3, but at the end of the dictionary attributes three bytes are appended: either the length of the item this item is subordinate to, or the VLC of this item.		

Figure 65 (Part 1 of 2). LD Entry Variable Information

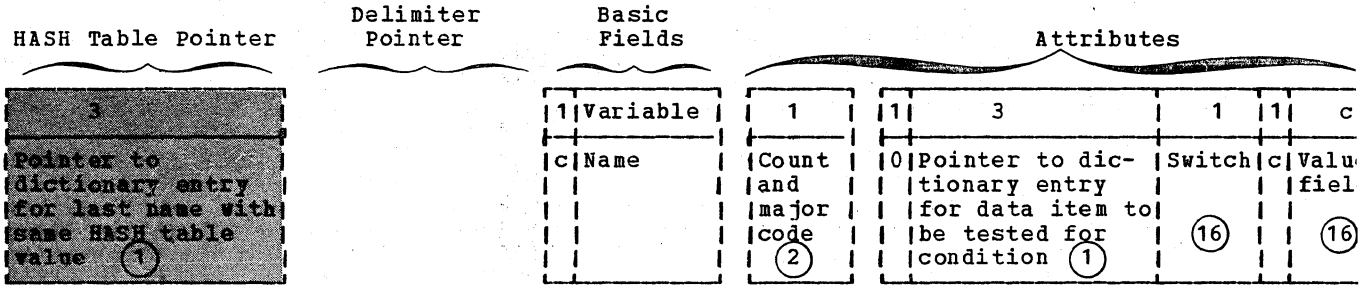
Characteristic	Bits	Contents
Variable-length Group Item	1-2	Unused.
	3-5	Number of BLs or BLLs for items.
	6-17	VLC number.
	18-32*	Dictionary section number of entry for group in which this item is included, if the group contains an OCCURS clause.
	33-41*	Displacement in section.
Alphanumeric Edited Item	1	1, if JUSTIFIED RIGHT
	2-9	Number of bytes following.
	10-17**	All zeros.
	18-32*	Dictionary section number of entry for group in which this item is included, if the group contains an OCCURS clause.
	33-41*	Displacement in section.
	42-57	SIZE of item.
	58 to end	Byte 1 contains PICTURE character. Bytes 2 and 3 contain count of consecutive occurrences. These three bytes are repeated until the entire PICTURE is recorded.
Elementary Item with Report PICTURE	1	1, if Z or * in PICTURE
	2-9	Number of bytes following.
	10-17**	All zeros.
	18-32*	Dictionary section number of entry for group in which this item is included, if the group contains an OCCURS clause.
	33-41*	Displacement in section.
	42	1, if BLANK WHEN ZERO
	43	1, if * represents all numeric characters
	44	Unused.
	45-49	Number of digit places in item.
	50-57	Scaling factor.
	58-65	SIZE of item.
	66 to end	Byte 1 contains PICTURE character (except v or P). Byte 2 contains count of consecutive occurrences. These two bytes are repeated until the entire PICTURE is recorded. <u>Exception:</u> For CR and DB, the first character appears in byte 1, the second in byte 2.
	Elementary Item with Sterling Report PICTURE	1
2-9		Same as for Report PICTURE above.
10-17**		All zeros.
18-32*		Dictionary section number of entry for group in which this item is included, if the group contains an OCCURS clause.
33-41*		Displacement in section.
42		1, if BLANK WHEN ZERO.
43		0, if shilling delimiter is D. 1, if shilling delimiter is S.
44		Same as bit 43 for pounds.
45		1, if no pounds field.
46-57		Unused.
58-65		Total length of item.
66-73	Number of pound integer places.	
74-81	Number of pence decimal places.	

\*Entry contains these fields only if item is in a group with an OCCURS clause. The fields are not present after phase 3, but at the end of the dictionary attributes three bytes are appended: either the length of the item to which this item is subordinate, or the VLC of this item.

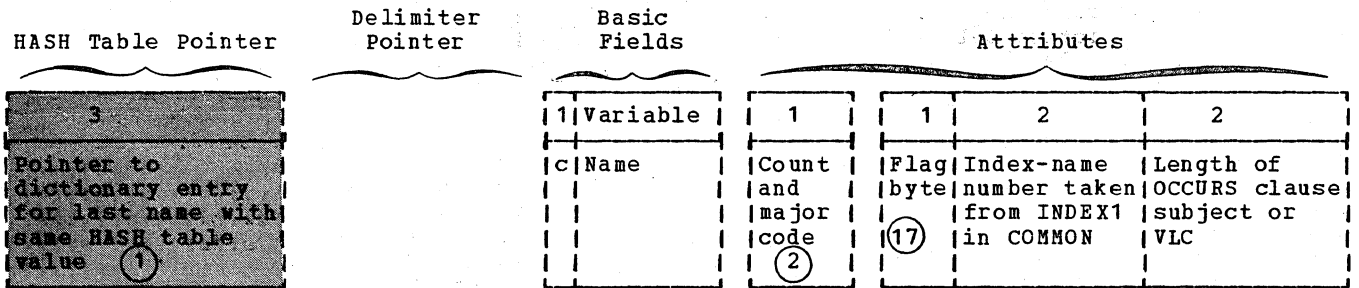
\*\*Entry does not contain these fields after phase 3.

Figure 65 (Part 2 of 2). LD Entry Variable Information

CONDITION-NAME ENTRY



INDEX-NAME ENTRY



(16) If the switch = 1, the value field contains a 2-byte displacement in the VALTRU table of the object of the VALUE clause. If the switch = 0, the field contains the value itself.

(17) Bit Meaning, if on  
 0 Subject is variable length; last field contains VLC number.  
 1-7 Unused.

DEBUG DATA SET TABLES

When the SYMDMP, STATE, or TEST option is in effect, phases 25 and 65 build additional tables for debugging purposes.

The tables are accessed during execution of the program or at abnormal termination of the program by the subroutines of the Symbolic Debug program. For details see the publication IBM OS/VS COBOL Subroutine Library Program Logic.

The tables list the characteristics of the data areas defined by the user as well as information about the relative location in the object module of the code associated with the card numbers generated for PNs and COBOL verbs. This information is used by the object-time COBOL library subroutines to produce the dumps at user-specified card numbers or card and verb numbers or by the COBOL Interactive Debug Program. If abnormal termination occurs, this information is also used to associate the address of the instruction at which abnormal termination occurred with its corresponding card and verb number in the COBOL source program.

The tables are also used by the IBM OS COBOL Interactive Debug Program Product (Program Number 5734-CB4).

Phase 25 builds the OBODOTAB table if there are any OCCURS clauses with the DEPENDING ON option. It also builds the DATATAB table. Phase 65 builds the PROGSUM, PROCTAB, CARDINDX, SEGINDX, PROCINDX and BCDPN tables.

The tables are made up of fixed-length 512-byte blocks; a 1-byte field containing the hexadecimal value 'FF' marks the end of

usable information within a block. Table entries are never split across a block.

The debug data set is single buffered and the address of the buffer is placed by phase 02 in location FIL5BUF in COMMON. Each phase that uses the debug data set is responsible for moving information into the buffer and marking the end of the buffer. Phase 00 is called only to write the buffer on the debug data set.

Figure 66 shows the positions of the tables in the debug data set (SYSUT5). The PROGSUM table contains information about, and pointers to, the other tables in the data set.

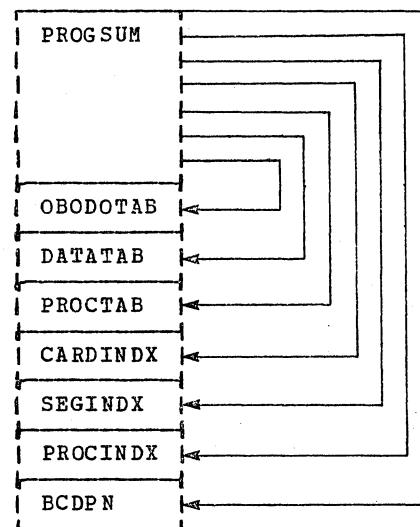


Figure 66. SYSUT5 (Debug Data Set)

PROGSUM TABLE

The PROGSUM table is the first table in a debug data set. It consists of a single fixed-length 108-byte entry and contains information about the program and the debug data set itself.

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>
<u>Hex</u>	<u>Decimal</u>			
0	0	PGPROGID	8	PROGRAM-ID
8	8	PGDECLEN	4	Length of Declaratives Section
C	12	PGBL1	4	BL1 address relative to the start of the TGT
10	16	PGBLL1	4	BLL1 address relative to the start of the TGT
14	20	PGSBL1	4	SBL1 address relative to the start of the TGT
18	24	PGDECB1	4	DECB1 address relative to the start of the TGT
1C	28	PGVLC1	4	VLC1 address relative to the start of the TGT
20	32	PGINDX1	4	INDEX1 address relative to the start of the TGT
24	36	PGDCB1	4	DCB1 address in PGT
28	40	PGENDDCB	4	End of the DECBs relative to the start of the TGT
2C	44	PGENDNDX	4	End of the indexes relative to the start of the TGT
30	48	PGD TDVAD	4	Device address of first block in DATATAB
34	52	PGDTNUM	2	Number of blocks in DATATAB
36	54	PGD TDSP	2	Displacement in the block of the first DATATAB entry
38	56	PGPTDVAD	4	Device address of PROCTAB
3C	60	PGDXDVAD	4	Device address of CARDINDX
40	64	PGSXDVAD	4	Device address of SEGINDX
44	68	PGPXDVAD	4	Device address of PROCINDX
48	72	PGCXNUM	2	Number of entries in CARDINDX
4A	74	PGSXNUM	2	Number of entries in SEGINDX
4C	76	PGPXNUM	2	Number of entries in PROCINDX
4E	78	PGSXDSP	2	Displacement in the block of the first SEGINDX entry
50	80	PGPXDSP	2	Displacement in the block of the first PROCINDX entry
52	82	PGODONUM	2	Number of bytes in OBODOTAB including the unused bytes at the end of the blocks
54	84	PGHASH	2	Hashed compilation indicator which is matched by the COBOL library subroutine with the one in the Debug Table in the TGT.
56	86	PGNUM	2	Number of blocks of BCDPNTBL
58	88	-	4	Reserved
5C	92	PGPNDVAD	4	Device address of first block of BCDPNTBL
60	96	PGTEFDSP	2	Displacement from beginning of PGT to ILBOTEF3 virtual if present
62	98	PGSMDSP	2	Unused
64	100	PGNBCD	2	Number of entries in BCDPNTBL
66	102	PGPH25W	2	Number of blocks written by phase 25
68	104	PGLNGTH	1	Length of PROGSUM
69	105	-	3	Unused
6C	108	PGFIB	4	Address of first File Information Block (FIB) relative to the start of the TGT.

**Note:** The only fields that may be zero in this table are PGDECLEN, when there is no Declarative Section and PGODONUM, when there are no OCCURS...DEPENDING ON clauses and PGTEFDSP and PGSMDSP when the corresponding virtuals are not present. For TGT addresses which do not exist, the address of the first byte following the previous cell is used because these cells are used in calculating the number of TGT cells of a given kind to dump.



OBODOTAB TABLE

The OBODOTAB table is an abstract of the DATATAB entries for all objects of OCCURS...DEPENDING ON clauses in the program. The OBODOTAB table, if present, follows on the next fullword boundary the PROGSUM table and contains one variable-length entry for each unique object of an OCCURS...DEPENDING ON clause. Each entry begins on a fullword boundary within the block.

The entries are essentially the same as the DATATAB entries for the same name. See the entries for elementary numeric items in the format of the DATATAB table. OBODOTAB entries differ only in that the card-number field is zero and the renaming and subscripting information is omitted. Table-locators within the DATATAB entries are used to access the OBODOTAB entries. See the subscripting information portion in the format of the DATATAB table.

COUNT-NAME-TYPE FIELD

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>
<u>Hex</u>	<u>Decimal</u>			
0	0		1	Count field: number of bytes (c) in name field
1	1		c	Name field: number of bytes, varies between 1 and 30
1+c	1+c		1	Count field: number of bytes in remainder of this entry
2+c	2+c	CARDNUM	3	Card number where name is defined
5+c	5+c	MAJMIN	1	Type of entry (For description of this field see corresponding field in DATATAB table)

VARIABLE ATTRIBUTES FIELD

For description of this field see corresponding field in DATATAB table.

DATATAB TABLE

The DATATAB table is the third table in the debug data set. It immediately follows the last entry of the OBODOTAB table, if that table is present. Otherwise, it follows the PROGSUM table. The DATATAB table lists the characteristics of each data item in the Data Division. The table consists of two fields, the Count-Name-Type field and the Variable Attributes field. The Count-Name-Type field has the same format for all entries. It varies in length from 7 to 36 bytes. The Variable Attributes field differs for each type of entry and is described on the following pages.

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>COUNT-NAME-TYPE FIELD</u>	
<u>Hex</u>	<u>Decimal</u>			<u>Description</u>	
0	0		1	Count field:	number of bytes (c) in name field
1	1		c	Name field:	number of bytes varies between 1 and 30
1+c	1+c		1	Displacement field:	Number of bytes from this field to TEST field (zeros if TEST field is not present)
2+c	2+c		* 1	Count field:	number of bytes in remainder of entry
3+c	3+c	CARDNUM	3	Card number where name is defined	(contains zeros for RENAMES items)
6+c	6+c	MAJMIN	1	Type of entry	

<u>Bits</u>	<u>Settings</u>	<u>Meaning</u>
0-3	0000XXXX	Level description under FD
	0001XXXX	Level Description under SD
	0010XXXX	Level description under CD
	0100XXXX	Level description in Working-Storage
	0101XXXX	Level description in Linkage
	0110XXXX	Level Description under RD
	1000XXXX	FD entry (other than VSAM)
	1001XXXX	SD entry
	1010XXXX	CD entry
	1011XXXX	FD entry (VSAM)
	XXXX0001	Organization sequential
	XXXX0010	RD entry
	XXXX0011	Index-name
	4-7	XXXX0001
XXXX0010		Alphabetic
XXXX0011		Alphanumeric
XXXX0100		Variable-length group
XXXX0101		Numeric edited
XXXX0110		Sterling report
XXXX0111		Usage index
XXXX1000		External decimal
XXXX1001		External floating-point
XXXX1010		Internal floating-point
4-7	XXXX1011	Binary
	XXXX1100	Internal decimal
	XXXX1101	Sterling nonreport
	XXXX1110	Alphanumeric edited
	XXXX1111	RENAMES (level 66)

\* Note 1: Number of bytes includes the byte field itself, if number of entries is not equal to zero.

SD Item: There are no variable attributes for an SD entry.

RENAMES item (level 66):

VARIABLE ATTRIBUTES FIELD

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>		
<u>Hex</u>	<u>Decimal</u>			<u>Bit</u>	<u>Setting</u>	<u>Meaning</u>
7+c	7+c	RENAMES	1	7	XXXXXXX1	Next DATATAB entry renames the same item as this one does
					XXXXXXX0	This is the last (or only) item renaming an item.
8+c	8+c		variable			Field contents are the same as the type-dependent portion of the level description item for the particular data type of the RENAMES item.

INDEX name:

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>		
<u>Hex</u>	<u>Decimal</u>			<u>Bit</u>	<u>Setting</u>	<u>Meaning</u>
7+c	7+c	INDFLAG	1			Flag
8+c	8+c	INDXCELL	2	0	1	Subject of OCCURS clause is variable length, next field contains VLC number
				1-7		Unused
A+c	10+c	INDLNG	2			Index cell number in TGT Length of subject of OCCURS clause or VLC number

FD Item (VSAM):

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>		
<u>Hex</u>	<u>Decimal</u>			<u>Bits</u>	<u>Setting</u>	<u>Meaning</u>
7+c	7+c	FIB	1			FIB number
8+c	8+c	ORGACC	1			ORGANIZATION and ACCESS clauses
				0-3	1000XXXX	Access sequential
					0100XXXX	Access random
					0010XXXX	Access dynamic
				6-7	XXXX0100	Organization relative
					XXXX0010	Organization indexed
					XXXX0001	Organization sequential

FD Item (other than VSAM):

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>		
<u>Hex</u>	<u>Decimal</u>			<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>
7+c	7+c	DCBDECB	1			DCB or DECB number
8+c	8+c	ACCESSFLG	1			Access method
				0-3	0000XXXX	QSAM
					0001XXXX	QISAM
					0010XXXX	BISAM
					0100XXXX	BSAM
					1000XXXX	BDAM
				7	XXXXXXX1	DECB Number in preceding byte
					XXXXXXX0	DCB Number in preceding byte

CD Item:

<u>Displacement</u>		<u>Description</u>		
<u>Hex</u>	<u>Decimal</u>	<u>Bit</u>	<u>Setting</u>	<u>Meaning</u>
7+c	7+c		1XXXXXXXX	Output
			0XXXXXXXX	Input

RD Item:

7+c	7+c	LINECTR	3	Addressing parameters of line counter; contains zeros if line counter is not defined																		
				<table border="0"> <thead> <tr> <th><u>Bit</u></th> <th><u>Settings</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>0-3</td> <td>0000XXXX</td> <td>BL entry</td> </tr> <tr> <td></td> <td>0001XXXX</td> <td>BLL entry</td> </tr> <tr> <td></td> <td>0100XXXX</td> <td>SBL entry</td> </tr> <tr> <td>4-15</td> <td></td> <td>Displacement from BL</td> </tr> <tr> <td>16-23</td> <td></td> <td>BL Number</td> </tr> </tbody> </table>	<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>	0-3	0000XXXX	BL entry		0001XXXX	BLL entry		0100XXXX	SBL entry	4-15		Displacement from BL	16-23		BL Number
<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>																				
0-3	0000XXXX	BL entry																				
	0001XXXX	BLL entry																				
	0100XXXX	SBL entry																				
4-15		Displacement from BL																				
16-23		BL Number																				
A+c	10+c	PAGECTR	3	Addressing parameters of page counter (same form as addressing parameters above), contains zeros if page counter is not defined																		

Level Description item:

Variable attributes for level description items are divided into two portions: (1) the type-dependent portion, and (2) subscripting information portion. The subscripting information portion is the same for all level description item entries. It follows and is described after the type-dependent portion descriptions.

(1) Type-dependent portion:

FIXED-LENGTH GROUP:

7+c	7+c	IDKFLD	3	Addressing parameters (same form as above)												
A+c	10+c	LVLRDEFN	3													
				<table border="0"> <thead> <tr> <th><u>Bit</u></th> <th><u>Settings</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>0-5</td> <td>XXXXXX1X</td> <td>Normalized level number</td> </tr> <tr> <td>6</td> <td></td> <td>REDEFINES</td> </tr> <tr> <td>7-23</td> <td></td> <td>Object-time storage size (in bytes)</td> </tr> </tbody> </table>	<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>	0-5	XXXXXX1X	Normalized level number	6		REDEFINES	7-23		Object-time storage size (in bytes)
<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>														
0-5	XXXXXX1X	Normalized level number														
6		REDEFINES														
7-23		Object-time storage size (in bytes)														

VARIABLE-LENGTH GROUP:

7+c	7+c	MAXSIZE	3	Addressing parameters (same form as above)												
A+c	10+c															
				<table border="0"> <thead> <tr> <th><u>Bit</u></th> <th><u>Settings</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>0-5</td> <td>XXXXXX1X</td> <td>Normalized level number</td> </tr> <tr> <td>6</td> <td></td> <td>REDEFINES</td> </tr> <tr> <td>7-23</td> <td></td> <td>Maximum object-time storage size (in bytes)</td> </tr> </tbody> </table>	<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>	0-5	XXXXXX1X	Normalized level number	6		REDEFINES	7-23		Maximum object-time storage size (in bytes)
<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>														
0-5	XXXXXX1X	Normalized level number														
6		REDEFINES														
7-23		Maximum object-time storage size (in bytes)														

D+c	13+c	VLCNUM	2	<table border="0"> <thead> <tr> <th><u>Bit</u></th> <th><u>Settings</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1XXXXXXX</td> <td>ODO master</td> </tr> <tr> <td>1-3</td> <td></td> <td>Unused</td> </tr> <tr> <td>4-15</td> <td></td> <td>VLC number</td> </tr> </tbody> </table>	<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>	0	1XXXXXXX	ODO master	1-3		Unused	4-15		VLC number
<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>														
0	1XXXXXXX	ODO master														
1-3		Unused														
4-15		VLC number														

ELEMENTARY, ALPHABETIC, ALPHANUMERIC, REPORT, EDITED, STERLING, EXTERNAL FLOATING-POINT:

7+c	7+c	JSTRGT	3	Addressing parameters (same form as above)															
A+c	10+c																		
				<table border="0"> <thead> <tr> <th><u>Bit</u></th> <th><u>Settings</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>0-5</td> <td></td> <td>Normalized level number</td> </tr> <tr> <td>6</td> <td>XXXXXX1X</td> <td>REDEFINES</td> </tr> <tr> <td>7</td> <td>XXXXXXX1</td> <td>JUSTIFIED RIGHT</td> </tr> <tr> <td>8-23</td> <td></td> <td>Object-time storage size (in bytes)</td> </tr> </tbody> </table>	<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>	0-5		Normalized level number	6	XXXXXX1X	REDEFINES	7	XXXXXXX1	JUSTIFIED RIGHT	8-23		Object-time storage size (in bytes)
<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>																	
0-5		Normalized level number																	
6	XXXXXX1X	REDEFINES																	
7	XXXXXXX1	JUSTIFIED RIGHT																	
8-23		Object-time storage size (in bytes)																	

INTERNAL FLOATING-POINT:

7+c	7+c		3	Addressing parameters (same form as above)
A+c	10+c	FLPTYPE	1	

Bit	Settings	Meaning
0-5		Normalized level number
6	XXXXXX1X	REDEFINES
7	XXXXXX0	COMP-1
	XXXXXX1	COMP-2

B+c	11+c		2	Unused
-----	------	--	---	--------

BINARY, INDEX INTERNAL DECIMAL, EXTERNAL DECIMAL:

7+c	7+c		3	Addressing parameters (same form as above)
A+c	10+c	NUMINF01	1	

Bit	Settings	Meaning
0-5		Normalized level number
6	XXXXXX1X	REDEFINES
7	XXXXXX1	S in PICTURE
0	1XXXXXX	Leading sign
	0XXXXXX	Trailing sign
1	X1XXXXXX	Separate sign
	X0XXXXXX	Overpunch
2	XX1XXXXX	Significant digits left of decimal point
	XX0XXXXX	No significant digits left of decimal point
3	XXX1XXXX	Significant digits right of decimal point
	XXX0XXXX	No significant digits right of decimal point
4-8		If bit 2 equals 1, number of digits to left of decimal point. If bit 2 equals 0, number of digits to right of decimal point.
9-13		If bits 2 and 3 both equal 1, number of digits to right of decimal point. If only bit 2 or 3 equals 1, number of Ps in picture
14-15		Unused

B+c	11+c	NUMINF02	2	
-----	------	----------	---	--

(2) Subscripting Information Portion:

This portion of the Variable Attributes field begins immediately after the type-dependent portion.

It ranges in size from 2 bytes for an unsubscripted item to a maximum of 20 bytes for an item belonging to 3 variable-length groups.

1 Guide to RENAMES and subscripting

<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>
0	1XXXXXXX	This item is renamed. The next DATATAB entry renames it.
1	X1XXXXXX	This item contains an ODO clause.
2	XX1XXXXX	Item requires at least 1 subscript.
3	XXX1XXXX	OCCURS clause connected with the most inclusive or only group; or elementary item contains an ODO.
4	XXXX1XXX	Item requires at least two subscripts.
5	XXXXX1XX	OCCURS clause connected with the less inclusive group of 2 groups or the middle inclusive group of 3 groups or elementary group contains an ODO.
6	XXXXXX1X	Item requires 3 subscripts.
7	XXXXXXX1	OCCURS clause connected with the least inclusive group of 3 groups or elementary item contains an ODO.

1 VLC information

<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>
0	1XXXXXXX	Most inclusive of 3 groups or only group
1	X1XXXXXX	Less inclusive group of 2 groups or middle inclusive group of 3 groups
2	XX1XXXXX	Least inclusive group of 3 groups
3-6		Unused
7	XXXXXXX1	This item contains an object of an ODO or is the object of an ODO clause

If bits 0, 1, or 2 equals 1, the displacement of next occurrence field for the associated group contains a VLC number (see note 2).

1st subscript (if present)	2	Number of occurrences (maximum number if ODO) specified in OCCURS clause governing this item						
(most inclusive with OCCURS)	2	Displacement of next occurrence governed by OCCURS clause (see note 2)						
2nd subscript (if present)	2	Number of occurrences (as above)						
	2	Displacement of next occurrence governed by OCCURS clause						
3rd subscript (if present)	2	Number of occurrences (as above)						
(least inclusive with OCCURS)	2	Displacement of next occurrence governed by OCCURS clause						
1st subscript with ODO (if present)	2	OBODOTAB pointer for most inclusive group or elementary item containing an ODO						
		<table border="0"> <tr> <td style="padding-right: 20px;"><u>Bit</u></td> <td><u>Contents</u></td> </tr> <tr> <td>0-8</td> <td>Relative block number in OBODOTAB</td> </tr> <tr> <td>9-15</td> <td>Displacement with block (in fullwords)</td> </tr> </table>	<u>Bit</u>	<u>Contents</u>	0-8	Relative block number in OBODOTAB	9-15	Displacement with block (in fullwords)
<u>Bit</u>	<u>Contents</u>							
0-8	Relative block number in OBODOTAB							
9-15	Displacement with block (in fullwords)							
2nd subscript with ODO (if present)	2	OBODOTAB pointer for less inclusive group (as above)						
3rd subscript with ODO (if present)	2	OBODOTAB pointer for least inclusive group (as above)						

(3) TEST Field:

This portion immediately follows the subscripting information and is pointed to by the displacement field in the COUNT-NAME-TYPE portion. This field is present only for the following data types when the TEST option is in effect:

<u>External Floating-point:</u>	<u>Bytes</u>	<u>Description</u>												
	1	Count of bytes following												
	2	<table border="0"> <tr> <td style="padding-right: 20px;"><u>Bit</u></td> <td><u>Meaning</u></td> </tr> <tr> <td>0-6</td> <td>Unused</td> </tr> <tr> <td>7</td> <td>0 = mantissa blank when positive 1 = sign plus when positive</td> </tr> <tr> <td>8</td> <td>0 = exponent blank when positive 1 = sign plus when positive</td> </tr> <tr> <td>9</td> <td>0 = implied decimal point 1 = real decimal point</td> </tr> <tr> <td>10-15</td> <td>Scale of mantissa</td> </tr> </table>	<u>Bit</u>	<u>Meaning</u>	0-6	Unused	7	0 = mantissa blank when positive 1 = sign plus when positive	8	0 = exponent blank when positive 1 = sign plus when positive	9	0 = implied decimal point 1 = real decimal point	10-15	Scale of mantissa
<u>Bit</u>	<u>Meaning</u>													
0-6	Unused													
7	0 = mantissa blank when positive 1 = sign plus when positive													
8	0 = exponent blank when positive 1 = sign plus when positive													
9	0 = implied decimal point 1 = real decimal point													
10-15	Scale of mantissa													
<u>ALPHANUMERIC EDITED:</u>	1	Count of bytes following												
	1	PICTURE character												
	2	Count of consecutive occurrences												
	n+3	The preceding three bytes are repeated until the entire PICTURE is recorded												

NUMERIC EDITED:

1	Count of bytes following
2	<u>Bit</u> <u>Meaning</u>
	0      1 = Blank when zero
	1      1 = * represents all numeric characters
	2      1 = Z or * in PICTURE
	3-7    Number of digit places in item
	8-15   Scaling factor
2	0-7    Contains PICTURE character (except V or P)
	8-15   Contains count of consecutive occurrences
N+2	These bytes are repeated until the entire PICTURE is recorded

Note 1: For CR or DB, first character appears in byte 1, the second in byte 2.

Note 2: All subscript length information precedes any OBODOTAB pointers. If the OCCURS clause is applicable, the contents of the displacement field is as follows:

1. For an elementary item, it is the machine length of that item including any slack bytes that precede its next occurrence.
2. For a fixed-length group, it is the length of the group including any slack bytes added at the end of the group before its next occurrence.
3. For a variable-length group, it is the VLC number for the VLC field which contains the number of bytes to the next occurrence.



PROCTAB TABLE

The PROCTAB table contains one 5-byte entry for each card and/or verb in the source listing of the COBOL Procedure Division. The table is ordered on three levels:

1. Priority (in ascending order of independent segments, with the root segment last)
2. Card-number within priority
3. Verb-number within card

The last PROCTAB entry for a priority and for a program fragment has a card and verb number of zero. In addition, the relative address field contains the address of the first byte following all instructions for the segment with that priority.

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>						
<u>Hex</u>	<u>Decimal</u>									
0	0	PTCDVB	3	Card number and verb number on source listing						
				<table border="1"> <thead> <tr> <th><u>Bit</u></th> <th><u>Contents</u></th> </tr> </thead> <tbody> <tr> <td>0-19</td> <td>Card number</td> </tr> <tr> <td>20-23</td> <td>Verb number</td> </tr> </tbody> </table>	<u>Bit</u>	<u>Contents</u>	0-19	Card number	20-23	Verb number
<u>Bit</u>	<u>Contents</u>									
0-19	Card number									
20-23	Verb number									
3	3	PTRELAD	2	Relative address of instructions for this entry within program fragment to which it belongs						

CARDINDX TABLE

The CARDINDX table is a directory to the SEGINDX table and contains one 5-byte entry for each program fragment and one entry for each discontinuity in the COBOL instructions within a segment. Entries in the CARDINDX table are in ascending card number order and are accessed by indexing through the table sequentially.

The CARDINDX table starts at the beginning of a block.

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>						
<u>Hex</u>	<u>Decimal</u>									
0	0	CXCDVB	3	Card number and verb number of first card represented by this entry						
				<table border="1"> <thead> <tr> <th><u>Bit</u></th> <th><u>Contents</u></th> </tr> </thead> <tbody> <tr> <td>0-19</td> <td>Card number</td> </tr> <tr> <td>20-23</td> <td>Verb number</td> </tr> </tbody> </table>	<u>Bit</u>	<u>Contents</u>	0-19	Card number	20-23	Verb number
<u>Bit</u>	<u>Contents</u>									
0-19	Card number									
20-23	Verb number									
3	3	CXPRIOR	1	Priority number associated with this card						
4	4	CXFRAG	1	Relative fragment number within the priority to which this card belongs						

SEGINDX TABLE

The SEGINDX table contains one 10-byte entry for each program fragment. The table is ordered on ascending fragment number:

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>
<u>Hex</u>	<u>Decimal</u>			
0	0	SXPRIOR	1	Zero
1	1	SXRELAD	3	Address of this fragment relative to the beginning of the program
4	4	SXPTLOC1	3	Table locator for PROCTAB entry of first card number and verb number in this fragment
				<u>Bit</u> <u>Contents</u>
				0-14      Relative block number in PROCTAB
				15-23     Displacement within block
7	7	SXPTLOC2	3	Table locator for PROCTAB entry of last card and/or verb in this fragment

PROCINDX TABLE

The PROCINDX table is a summary index of the PROCTAB table and contains one 10-byte entry for each block of PROCTAB entries. PROCINDX entries are ordered by relative block number in the PROCTAB table and are accessed by searching sequentially after indexing to a starting point determined by the block number from the SEGINDX table.

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>
<u>Hex</u>	<u>Decimal</u>			
0	0	PXCDVB	3	Card number and verb number of first entry in block of PROCTAB table.
				<u>Bit</u> <u>Contents</u>
				0-19      Card number
				20-23     Verb number
3	3	PXRELAD	3	Address of instructions for this entry relative to the beginning of the program.
6	6	PXDEVADR	4	Device address of PROCTAB table block related to this entry.

BCDPN TABLE

The BCDPN table is a list of EBCDIC PN names and their corresponding line-numbers. It has 1 entry for each PN in the COBOL program. Its maximum length is 27 bytes. It begins at the beginning of a block.

<u>Displacement</u>		<u>Field Name</u>	<u>Bytes</u>	<u>Description</u>
<u>Dec</u>	<u>Hex</u>			
0	0	PNLINPNO	3	Line number of PN
3	3	PNNUMBYT	1	Number of bytes following
4	4	PNNAME	Var.	Compressed PN name

VSAM FILE INFORMATION BLOCK (FIB)

The file information block, a portion of the completed object module, is used at execution time by the ILBOINTO, ILBOVOCO, and ILBOVIOO COBOL library subroutine for processing input/output verbs used with VSAM files. The FIB is built by phases 21 and completed by the ILBOVOL0 subroutine.

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>
<u>Hex</u>	<u>Decimal</u>			
0	0	IF1BID	1	FIB identification code ('I')
1	1	IF1BLVL	1	FIB level number

Fixed Portion:

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>
<u>Hex</u>	<u>Decimal</u>			
2	2	INAMED	7	External name
9	9	INAMECB	1	External name
A	10	IDEVICE	1	Device class and number
B	11	IORG	1	ORGANIZATION

Code:

<u>Bits</u>	<u>Equate Name</u>	<u>Bit Settings</u>	<u>Meaning</u>	<u>Source</u>
0-7	IORVPS	1000 1000	VSAM ADDRESSED SEQUENTIAL	Code=A
	IORGSQO	1000 0100	SEQUENTIAL	Code=S
	IORGSQN	1000 0010	SEQUENTIAL	
	IORGASC	1000 0001	ASCII file SEQUENTIAL	Code=C
	IORGSEQ	1000 0000	SEQUENTIAL	ORGANIZATION IS SEQUENTIAL
	IORGVIX	0100 1000	VSAM INDEXED	
	IORGINO	0100 0100	INDEXED	Code=I
	IORGIND	0100 0000	INDEXED	ORGANIZATION IS INDEXED
	IORGVRL	0010 1000	VSAM RELATIVE	
	IORGRLO	0010 0100	RELATIVE	Code=R
	IORGDIR	0010 0010	DIRECT	Code=D
	IORGDIW	0010 0001	DIRECT (WRITE/REWRITE)	Code=W
	IORGREL	0010 0000	RELATIVE	ORGANIZATION IS RELATIVE
	IORGVSAM	0000 1000	VSAM File	

C	12	IACCESS	1	ACCESS MODE
---	----	---------	---	-------------

Code:

<u>Bits</u>	<u>Equate Name</u>	<u>Bit Settings</u>	<u>Meaning</u>
0-7	IACCESQ	1000 0000	SEQUENTIAL
	IACCRAN	0100 0000	RANDOM
	IACCDYN	0010 0000	DYNAMIC

D	13	IRCDMODE	1	RECORDING MODE
---	----	----------	---	----------------

Code:

<u>Bits</u>	<u>Equate Name</u>	<u>Bit Settings</u>	<u>Meaning</u>
0-7	IRCDFIX	1000 0000	FIXED
	IRCDVAR	0100 0000	VARIABLE
	IRCDUND	0010 0000	Undefined
	IRCDSPN	0001 0000	SPANNED

Displacement		Field	No. of Bytes	Description
Hex	Decimal			
E	14	ISW1	1	Miscellaneous switches
Code:				
		<u>Bits</u>	<u>Equate Name</u>	<u>Bit Settings</u> <u>Meaning</u>
		0-7	ISOPTNL	1000 0000    OPTIONAL specified
			ISBLKED	0100 0000    File is blocked
			ISSAMREC	0010 0000    SAME RECORD AREA specified
			ISSAME	0001 0000    SAME RECORD specified
			ISLBOMIT	0000 1000    LABEL RECORDS ARE OMITTED
			ISLBSTAN	0000 0100    LABEL RECORDS ARE STANDARD
			ISLBUSER	0000 0010    LABEL RECORDS ARE dataname
F	15	ISW2	1	Miscellaneous switches
Code:				
		<u>Bits</u>	<u>Equate Name</u>	<u>Bit Settings</u> <u>Meaning</u>
		0-7	ISADVAN	1000 0000    WRITE ADVANCING
			ISPOSIT	0100 0000    WRITE POSITIONING
			ISAFTER	0010 0000    WRITE AFTER
			ISBEFORE	0001 0000    WRITE BEFORE
			ISNOSPAC	0000 1000    WRITE WITHOUT SPACING
10	16	ISW3	1	Unused
11	17	ISW4	1	Unused
12	18	IAPPLY1	1	APPLY statements
13	19	IAPPLY2	1	APPLY statements
14	20	IBLKLEN	2	If 'BLOCK CONTAINS (integer-1 TO) integer-2 CHARACTERS', field contains integer-2. If 'BLOCK CONTAINS (integer-1 TO) integer-2 RECORDS, field = integer-2 x (IRECLEN + control) + control + IASBFO Where control = 0 (Recording mode F or U) = 4 (Recording mode V, S, or D) IASBFO = 0 (Non-ASCII file) = Buffer offset (ASCII file) If BLOCK CONTAINS clause is omitted, field contents are same as for 'BLOCK CONTAINS 1 RECORD'.
16	22	IRECLEN	2	Number of bytes in longest 01-entry
18	24	IRECDBL	2	Displacement in TGT of record's first base locator cell
1A	26	IRECNBL	1	Number of base locators for RECORD AREA
1B	27	IRESERVE	1	Reserve integer areas
1C	28	ISTATDBL	2	Displacement in TGT of base locator for STATUS data-name
1E	30	ISTATDDN	2	Displacement from base locator of STATUS data-name
20	32	ISTATLDN	2	Length of STATUS data-name
22	34	IKEYISW	1	Miscellaneous switches
23	35	IKEYNO	1	Number of entries in key list
24	36	IKEYFNTL	2	Length of each entry in key list
26	38	IPSWISW	1	Miscellaneous switches
27	39	IPSWNO	1	Number of entries in password list
28	40	IPSWENTL	2	Length of each entry in password list
2A	42		14	Reserved
38	56	IMISCAD	4	Address in variable length portion of FIB for miscellaneous clauses
3C	60	ILABELAD	4	Address of labeling information block
40	64	IKEYLSTA	4	Address in variable length portion for first Key list entry
44	68	IPSWLSTA	4	Address in variable length portion of first password list entry.
48	72		16	

Variable Length Portion:

Supplementary Information for Miscellaneous Clauses (one for each clause):

<u>Displacement</u> <u>Hex</u> <u>Decimal</u>	<u>Field</u>	<u>No. of</u> <u>Bytes</u>	<u>Description</u>												
0        C	IMSW1	2	Switch bytes												
Code:															
			<table border="1"> <thead> <tr> <th><u>Bits</u></th> <th><u>Equate</u> <u>Name</u></th> <th><u>Bit</u> <u>Settings</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>0-7</td> <td>IMRREOV</td> <td>1000 0000</td> <td>RERUN at end of volume</td> </tr> <tr> <td>8-15</td> <td></td> <td></td> <td>Unused</td> </tr> </tbody> </table>	<u>Bits</u>	<u>Equate</u> <u>Name</u>	<u>Bit</u> <u>Settings</u>	<u>Meaning</u>	0-7	IMRREOV	1000 0000	RERUN at end of volume	8-15			Unused
<u>Bits</u>	<u>Equate</u> <u>Name</u>	<u>Bit</u> <u>Settings</u>	<u>Meaning</u>												
0-7	IMRREOV	1000 0000	RERUN at end of volume												
8-15			Unused												
2        2	IRERUNI	4	RERUN integer (Field contains zeros if RERUN not specified)												
6        6		2	Slack bytes												
8        8	IRERUNN	8	External-name of RERUN clause												

Indexed Record Key List:

0        0	IKEYSW	1	Miscellaneous switches																
Code    Equate    Bit																			
			<table border="1"> <thead> <tr> <th><u>Bits</u></th> <th><u>Name</u></th> <th><u>Settings</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>0-7</td> <td>IKEYCOMP</td> <td>1000 0000</td> <td>KEY is usage comp (binary)</td> </tr> <tr> <td></td> <td>IKEYDUP</td> <td>0100 0000</td> <td>WITH DUPLICATES specified</td> </tr> <tr> <td></td> <td>IKEYID</td> <td>0010 0000</td> <td>KEY is internal decimal</td> </tr> </tbody> </table>	<u>Bits</u>	<u>Name</u>	<u>Settings</u>	<u>Meaning</u>	0-7	IKEYCOMP	1000 0000	KEY is usage comp (binary)		IKEYDUP	0100 0000	WITH DUPLICATES specified		IKEYID	0010 0000	KEY is internal decimal
<u>Bits</u>	<u>Name</u>	<u>Settings</u>	<u>Meaning</u>																
0-7	IKEYCOMP	1000 0000	KEY is usage comp (binary)																
	IKEYDUP	0100 0000	WITH DUPLICATES specified																
	IKEYID	0010 0000	KEY is internal decimal																
1        1	IRKEYLDN	1	Length of RECORD KEY data name																
2        2	IKEYDBL	2	KEY is data names location, displacement in TGT.																
4        4	IRKEYDDN	2	Displacement in record of RECORD KEY Record. Key information follows for each ALTERNATE RECORD KEY specified.																

Parameter List:

0        0	IPSWDIXN	1	Associated index number 0 = none 1 = prime 2 = first alternate 3 = second alternate . . . n = (n-1)th alternate X'FF' = no password
1        1	IPSWDLDN	1	Length of password data-name.
2        2	IPSWDDBL	2	PASSWORD data name's locator; displacement in TGT.
4        4	IPSWDDDN	2	PASSWORD data name's displacement off locator.

SECTION 6. DIAGNOSTIC AIDS

This section contains aids for diagnosing errors that may have occurred in the compilation process. These aids are arranged under the following categories:

1. Procedure for applying the service aid program IMASPZAP in the event that certain disaster level error messages are generated by the compiler.
2. Compiler response to the findings of the system standard error recovery program.
3. Data set activity.
4. Register usage by each phase and execution-time register assignment.
5. Elements of program design that can help determine from a dump the stage of processing at which compilation has halted due to compiler error.

**PROCEDURE TO FORCE CORE DUMP**

In certain cases, the DUMP option will not result in a storage dump. Using the SPZAP service aid program, you can force a storage dump with the following procedure:

1. Locate the label TRMNATE in phase IKFCB100, CSECT PHOSECT2.
2. Verify that its contents is X'9620', then replace the contents with X'0000'.

More information on the SPZAP service aid program can be found in OS/VS Service Aids.

**RESPONSE TO SYSTEM ERROR RECOVERY FINDINGS**

Phases 00 and 02 interface with the system for compiler input/output operations and may, in the process, receive findings of the system error recovery program. In the case of phase 00, the SYNAD field of the DCB for each data set contains the address of a routine that generates an error message for that data set. The SYNAD routines associated with the data sets are as follows:

<u>Routine</u>	<u>Data Set</u>
SYAA	SYSLIB
SYAB	SYSUT1, SYSUT2, SYSUT3, SYSUT4, SYSUT5, SYSUT1 (dictionary spill), SYSIN, SYSPRINT (SYSUT6 for LVL option), SYSPUNCH, and SYSLIN
SYAD	SYSTEM

When an error of this type occurs, phase 00 terminates compilation immediately.

As part of its initialization operations phase 02 saves the exit list in each DCB (which may have been overridden by a DD card for OPEN statement processing. It inserts the addresses of its own routines, and then attempts to open each data set to test whether it is present and operative.

Routines starting at location EDIT check for such items as valid block size and insert default values if they are invalid. Routine OPENOK checks whether each data set can indeed be opened. Routine NODS1A then calls phase 00, placing a B in the X parameter when compilation is to be terminated.

Error messages are generated and placed in a table by routine QUE. When phase 02 has completed its processing, routine PRINT is called to write the message on SYSPRINT and SYSTEM if the TERM option is in effect or on SYSUT6 if the LVL option is in effect. Phase 02 then returns control to phase 01 which returns control to phase 00.

If neither SYSPRINT nor SYSTEM can be written on, messages are written on the console. If one of the two data sets can be written on, messages are written on that data set.

**COMPILER DATA SET ACTIVITY**

The compiler's use of data sets is shown in Figure 5 in the chapter "Phase 00." This table relates the phases and phase interludes to their requests that specific data sets be opened, written on, read from, or closed.

**REGISTER USAGE BY EACH PHASE**

The manner in which each phase uses registers is described in Figure 67.

Phase	Register	Use
00 (IKFCBL00)	0-1	Work.
	2	Address of data to be put out on any PUT (WRITE).
	3	Length of data for PUT (WRITE).
	4	Work. Base for PHOSECT2 CSECT.
	5	Base for IKFCBL00 CSECT.
	6	Points to buffer control block for logical input/output.
	7	Points to buffer control block for physical input/output.
	8	Return address minus 2 (linkage parameters).
	9	Address of file pointers (POINT TABLE) for the data set on which input/output is currently being performed.
	10	File number multiplied by 4.
	11	Linkage. Base register for TBCOMM.
	12	Linkage. Base register for START. Base register for TBDATA.
	13	LINK/RETURN information input/output request.
	14	Linkage. Work.
	15	Linkage. Work. Base for PHOTBST1.
01 (IKFCBL01)	0	Unused.
	1	Linkage to phase 02. Linkage to phase 00.
	2	Address of phase 00 parameter list.
	3-4	Unused.
	5	Base for CSECT IKFCBL01.
	6-12	Unused.
	13	Points to phase 01 save area.
	14-15	Unused.

Figure 67 (Part 1 of 12). Register Usage According to Phase

Phase	Register	Use	
02 (IKFCBL02)	0	Linkage to phase 00. Work.	
	1	Linkage to phase 00. Work.	
	2-4	Work.	
	5	Base for CSECT IKFCBL02.	
	6	Address of buffer control block.	
	7	Work.	
	8	Address of DCB.	
	9	Work.	
	10	Points to COMMON in phase 00.	
	11-12	Work.	
	13	Base for CSECT IKF021.	
	14	Internal linkage. Work.	
	15	Internal linkage. Work.	
	03 (IKFCBL03)	0-5	Work.
		6-10	Unused.
11		Base for instructions.	
12		Base for COS.	
13		Base for constants.	
14-15		Linkage.	
04 (IKFCBL04)	0-7	Work.	
	8-12	Base registers.	
	13-15	Save area's linkage.	

Figure 67 (Part 2 of 12) . Register Usage According to Phase



Phase	Register	Use	
05 (IKFCBL05)	0-3	Work.	
	4-8	Not used.	
	9	Base for COS/Common in phase 00.	
	10-13	Base for phase 05 SAVE-AREA, data areas, language tables, and all code.	
	14	Major linkage, work.	
	15	Subordinate linkage, work.	
	06 (IKFCBL06)	0	Work.
1		Returns most subroutine results, work.	
2		Pointers to or addresses of structure table entries, work.	
3		Work.	
4-5		Carry values over long spans, work.	
6-9		Not used.	
10		Base for current IP-text input item.	
11		Not used.	
12		Base for COS/Common in phase 00.	
13		Base for SAVE-AREA, data, and code.	
14		Linkage for all major subroutines.	
15		Linkage for other subroutines, contains arguments for some.	
08 (IKFCBL08)		0-3	Work.
		4-8	Not used.
		9	Base for COS in phase 00
	10-12	Base for first 12K of instructions and constants. (Value of R10 is 4096 greater than contents of R13.)	
	13	Base for system save area and declared data areas.	
	14	Linkage.	
	15	Explicit argument and/or function.	

Figure 67 (Part 3 of 12). Register Usage According to Phase

Phase	Register	Use																					
10 (IKPCBL10)	0	Work.																					
	1	Points to COMMON in phase 00.																					
	2-7	Work.																					
	8	During Data Division processing, base of CSECT IKF112 (FLUSH). At other times, work.																					
	9	During Data Division processing, base of CSECT IKF107 (LETTER).																					
	10	Permanent base for IKF104 (COBWRD).																					
	11	Permanent base for IKF103 (GETWD).																					
	12	Permanent base for IKF102 (IDDIV1).																					
	13	Permanent base for IKF101 (PH1SAV), pointing directly to the phase 10 save area in that CSECT.																					
	14	Linkage.																					
	15	Linkage to TAMER. Temporary base for subroutines:																					
			<table border="0"> <thead> <tr> <th><u>Used to address</u></th> <th><u>in CSECT</u></th> </tr> </thead> <tbody> <tr> <td>PROC01</td> <td>IKF113</td> </tr> <tr> <td>DDSCN</td> <td>IKF110</td> </tr> <tr> <td>ENVSCN</td> <td>IKF114</td> </tr> <tr> <td>LEVELQ</td> <td>IKF116</td> </tr> <tr> <td>FILCOQ</td> <td>IKF116</td> </tr> <tr> <td>SADSNQ</td> <td>IKF116</td> </tr> <tr> <td>IDDSCN</td> <td>IKF115</td> </tr> <tr> <td>UNLVSQ</td> <td>IKF106</td> </tr> <tr> <td>PRONAQ</td> <td>IKF108</td> </tr> </tbody> </table>	<u>Used to address</u>	<u>in CSECT</u>	PROC01	IKF113	DDSCN	IKF110	ENVSCN	IKF114	LEVELQ	IKF116	FILCOQ	IKF116	SADSNQ	IKF116	IDDSCN	IKF115	UNLVSQ	IKF106	PRONAQ	IKF108
	<u>Used to address</u>	<u>in CSECT</u>																					
	PROC01	IKF113																					
	DDSCN	IKF110																					
ENVSCN	IKF114																						
LEVELQ	IKF116																						
FILCOQ	IKF116																						
SADSNQ	IKF116																						
IDDSCN	IKF115																						
UNLVSQ	IKF106																						
PRONAQ	IKF108																						
12 (IKPCBL12)	0	Work.																					
	1	Points to COMMON in phase 00.																					
	2-7	Work.																					
	8	During Data Division processing, base of CSECT IKF112 (FLUSH). At other times, work.																					
	9	During Data Division processing, base of CSECT IKF111 (RDSCAN) or base of CSECT IKF107 (LETTER).																					
	10	Permanent base for IKF104 (COBWRD).																					
	11	Permanent base for IKF103 (GETWD).																					
	12	Permanent base for IKF102 (IDDIV1).																					

Figure 67 (Part 4 of 12). Register Usage According to Phase

Phase	Register	Use
12 (IKFCBL12) (continued)	13	Permanent base for IKF101 (PH1SAV), pointing directly to the phase 12 save area in that CSECT.
	14	Linkage.
	15	Linkage to TAMER.
		Temporary base for subroutines.
1B (IKFCBL1B)	0	Work.
	1	Points to COMMON in phase 00.
	2-8	Work.
	9	Base of IKF109 (EXHSVB). Base of IKF107 (LETTER).
	10	Base of IKF104 (COBWRD).
	11	Base of IKF103 (GETWD).
	12	Base of IKF102 (IDDIV1).
	13	Base of IKF101 (PH1SAV), pointing directly to the phase 1B save area in that CSECT.
	14	Linkage.
	15	Linkage to TAMER.
		Addressability to IKF107 (PDSCN) and temporary addressability to IKF109 (VARPQ) and to IKF106 (UNLVSN).
20 (IKFCBL20)	0-6	Work.
	7	Address of input buffer area.
	8-10	Work.
	11	Address of COMMON, except during ACCMET routine.
	12	Work.
	13	Pointer to save area.
	14	Branching.
	15	Base for all routines.

Figure 67 (Part 5 of 12). Register Usage According to Phase9

Phase	Register	Use
21 (IKFCBL21)	0-1	Work.
	2	BUFTAB pointer.
	3-6	Work.
	7	Data IC-text pointer
	8	Work.
	9	FDTAB pointer.
	10	PIOTBL pointer.
	11	Address of COMMON in phase 00.
	12	Base of PERMCODE.
	13	Pointer to save area.
	14-15	Linkage.
22 (IKFCBL22)	0-6	Work.
	7	Address of input buffer area.
	8-10	Work.
	11	Address of COMMON, except during ACCMET routine.
	12	Work.
	13	Pointer to save area.
	14	Branching.
	15	Base for all routines.
25 (IKFCBL25)	0-8	Work.
	9	Base for SYMDICT DSECT.
	10	Points to COMMON in phase 00.
	11	Permanent base for first CSECT IKF251.
	12	Permanent base for second CSECT IKF252.
	13	Permanent base for data CSECT IKF25A.
	14	Linkage.
	15	Linkage.
		Base for ACCESS routines.

Figure 67 (Part 6 of 12) . Register Usage According to Phase

Phase	Register	Use
3 (IKFCBL30)	0	Work.
	1	Frequently contains dictionary pointer to the entry currently being processed.
	2	Frequently points to the start of attributes in the dictionary entry currently being processed.
	3	Frequently contains dictionary pointer to the entry currently being processed.
	4-6	Work.
	7-10	Permanent bases for all routines and data.
	11-12	Rarely used (except by TAMER routines).
	13	Pointer to save area.
	14-15	Branching to ACCESS routines.
	35 (IKFCBL35)	0-10
11-12		Permanent bases for all routines.
13		Pointer to SAVE AREA, base for most of data area.
14		Branching; work.
15		Linkage; work.
4 (IKFCBL40)	0-6	Work; used by verb analyzers, subroutines, and phase controller. Register 5 also used by arithmetic routines to save input buffer pointer while scanning SETTBL.
	7-11	Permanent bases for the nonverb analyzer routines: phase controller, subroutines, and data area.
	12	Points to current element, either in input buffer or SETTBL.
	13	Unused.
	14	Branching.
	15	Temporary base register for the verb analyzer currently in control. Linkage to TAMER.

Figure 67 (Part 7 of 12). Register Usage According to Phase

Phase	Register	Use
45 (IKFCBL45)	0-1	Work.
	2-3	Parameter registers between internal subroutines.
	4-8	Work.
	9	Pointer to current text element (DOP).
	10	Points to COMMON in phase 00.
	11-12	Permanent base for control routine, verb analyzer, and subroutines.
	13	Permanent base for CSECT containing constants.
	14-15	Linkage.
50 (IKFCBL50)	0-6	Work.
	7	Base of first CSECT in the phase, beginning with PH5CTL.
	8	Base of second CSECT, beginning with XINSCN.
	9	Temporary base for verb analyzer routines beyond the third CSECT.
	10	Points to COMMON in phase 00.
	11	Base of third CSECT, beginning with A-text Generator.
	12	Base of phase 50 constant area (next-to-last CSECT).
	13	Base of phase 50 data area (last CSECT).
	14	Linkage.
	15	Temporary base for some routines called by verb analyzers.
51 (IKFCBL51)	0-6	Work.
	7	Base of first CSECT in the phase, beginning with PH5CTL.
	8	Base of second CSECT, beginning with XINSCN.
	9	Temporary base for verb analyzer routines beyond the third CSECT.
	10	Points to COMMON in phase 00.
	11	Base of third CSECT, beginning with A-text Generator.
	12	Base of phase 51 constant area (next-to-last CSECT).
	13	Base of phase 51 data area (last CSECT).
	14	Linkage.
15	Temporary base for some routines called by verb analyzers.	

Figure 67 (Part 8 of 12). Register Usage According to Phase

Phase	Register	Use
6 (IKFCBL60)	0-3	Work.
	4	Input for Optimization A-text.
	5-7	Work.
	8	Base for the following CSECTS: IKF601 -- Performs phase initialization (switches, work areas, etc.); IKF602, IKF603 -- Processing Listing A-text and Procedure A-text.
	9	Points to COMMON in phase 00.
	10	Base for the following CSECTS: IKF602 -- Processes Optimization A-text IKF604 -- Processes Procedure A-text and Data A-text; generates initialization routines.
	11	Base for IKF605 constants.
	12	Base for CSECT (IKF601 and IKF603).
	13	Points to phase 6 save area; base for CSECT containing constants (IKF606).
	14-15	Linkage to subroutines.
62 (IKFCBL62)	0-3	Work.
	4	Input for Optimization A-text and Procedure A-text.
	5-7	Work.
	8	Base for the following CSECTS: IKF62 -- Performs phase initialization (switches, work areas, etc.); processes the TGT and Optimization A-text. IKF623 -- Processes Procedure A-text
	9	Points to COMMON in phase 00
	10	Base for the following CSECTS: IKF62 -- Performs phase initialization (switches, work areas, etc.), processes the TGT and OPT A-text.
	11	Work.
	12	Base for CSECT containing constants IKF625.
	13	Points to phase 62 save area.
	14-15	Linkage to subroutines.

Figure 67 (Part 9 of 12). Register Usage According to Phase

Phase	Register	Use
63 (IKFCBL63)	0-10	Work.
	11	Address of COMMON in phase 00.
	12	Base for IKFCBL63.
	13	Save area and constants.
	14	Return address for internal subroutines.
	15	Address of internal subroutines.
64 (IKFCBL64)	0-3	Work.
	4	Input for Procedure A-text, Data A-text, and E-text.
	5-7	Work.
	8	Base for the following CSECTS: IKF64 -- Performs phase initialization (switches, work areas, etc) IKF643 -- Processes Procedure A-text
	9	Points to COMMON in phase 00.
	10	Base for the following CSECT: IKF644 -- Processes Procedure A-text and Data A-text; generates initialization routines.
	11	Work.
	12	Base for CSECT containing constants IKF645.
	13	Points to phase 64 save area (IKF6455).
	14-15	Linkage to subroutines.
	65 (IKFCBL65)	0-3
4		Input for Debug-text and for generating (TXTCRD) DSECT.
5-7		Work.
8		Base for CSECT IKF651.
9		Points to COMMON in phase 00.
10		Base for CSECT IKF652.
11		Work.
12		Points to PROCTAB entries.
13		Points to phase 65 save area; base for phase 65 constant CSECT IKF653.
14-15		Linkage to subroutines.

Figure 67 (Part 10 of 12). Register Usage According to Phase



Phase	Register	Use	
6A (IKFCBL61)	0	Work. Return address when phase 6A has branched to phase 00.	
	1	Work. Base for input DSECT.	
	2	Work. Pointer to data record being formatted.	
	3-9	Work.	
	10	Base for IKF6A02.	
	11	Points to COMMON in phase 00.	
	12	Base for CSECT IKF6A01.	
	13	Points to phase 6A save area (IKF6A01A).	
	14	Return address for internal subroutines.	
	15	Work. Address of internal subroutines.	
	70 (IKFCBL70)	0-4	Work.
		5	Points to message in message table during processing of most messages.
		6-7	Work.
		8	Base for phase instructions.
9		Points to COMMON in phase 00.	
10		Base for most constants.	
11-14		Work.	
15		Base for PUT, CONVERT, GET, STRING, and XPRIME routines.	

Figure 67 (Part 11 of 12). Register Usage According to Phase

Phase	Register	Use
80 (IKFCBL80-8D)	0	Work.
	1	Base for IKFCBL80. Work.
	2-3	DCB pointer for IKFCBL80. Work. Base for IKFCBL8C.
	4	DTF pointer for IKFCBL80. Base for IKFCBL8C. Work.
	5	Base for IKFCBL8C. Work.
	6	Internal link for IKFCBL8B, IKFCBL8C, and IKFCBL8D. Work.
	7-9	Work.
	10	DCB pointer for IKFCBL84, IKFCBL86, and IKFCBL8D.
	11	DCB pointer for IKFCBL8D. Work.
	12	Points to FIPSVT (FIPS vector table).
	13	Save.
	14	Link.
	15	Base.

Figure 67 (Part 12 of 12). Register Usage According to Phase

**REGISTER ASSIGNMENT**

The compiler assigns registers for use at execution time according to the general rules indicated in Figure 68. When the OPT option is in effect, the register assignment is that indicated in Figure 69. The DMAP, PMAP, or CLIST option causes the permanent register assignments for the data areas to be printed out along with the base locators associated with them.

Register	Assignment
0-5	Work
6	Pointer to beginning of Working-Storage
7-11	Pointers to FDs and then remainder of Working-Storage areas, if needed
12	Pointer to PGT
13	Pointer to TGT
14, 15	Temporary base registers

Figure 68. Register Usage at Execution

Register	Assignment
0-5	Work
6-9	Assigned in the following order: First, TGT or PGT OVERFLOW cells Then, most used BLs or BLLs
10	Assigned to PGT OVERFLOW cell if another OVERFLOW cell results from allocation of the PROCEDURE BLOCK cells or assigned to next most used BL or BLL
11	PROCEDURE BLOCK base register
12	Pointer to PGT
13	Pointer to TGT
14, 15	Temporary base registers

Figure 69. Register Usage at Execution (OPT)

**ELEMENTS OF PROGRAM DESIGN**

Elements in the design of the compiler can be used to determine a number of facts when analyzing a compiler error:

1. What conditions will cause the compiler itself to issue a message indicating compiler error?
2. Which phase is currently processing when an error terminates compilation, and which record is it processing?

3. Where are registers saved when one phase calls another, and what are their contents?
4. Where are the buffers that are being used, and what do they contain?
5. Where are the tables that are being used?

Compiler Error Messages

Certain error messages are issued only in case of compiler error. These messages take the following form:

IKFPnnnI-D COMPILER ERROR.

COMPILATION WILL NOT BE COMPLETE.

where p is the phase number and nnn is the message number. Figure 70 explains the conditions which cause these messages to occur.

**ABEND CODES**

If the DUMP option is specified, the following ABEND codes will appear (a description of the condition causing the ABEND, and the phase issuing the ABEND appear):

Phase	ABEND Code	Description	
00	000	SYSLIB SYNAD error	
	001	SYSTs files SYNAD error	
	002	System SYNAD error	
	006	Need larger size	
	007	DICOT entry not found for section. Spill request not met. TAMER request for non-printed table at TBL less than at MASTAM.	
	008	Table size over 32K	
	009	Fragmented core	
22	220	RENAMES error; occurs when IKFCBL22 is trying to calculate the length of an item with a RENAMES THRU clause and an error exists in the dictionary attributes in the characteristics of the object.	
	210	Unrecognizable input, text from IKFCBL22.	
30	312	Last item referenced by ACCESS was elementary.	
	313	Q-bit on, but pointer less than QVAR table pointer.	
	314	314	Q-bit on, but no match in QVAR table.

<u>Phase</u>	<u>ABEND Code</u>	<u>Description</u>			
	316	LATGRP return code 12 for CORRTR or end of table in INCRAV.	63	063	Error found processing input.
	317	Illegal minor code for data-name reference.	64	601	0x element, x ≠ 0, 1 or 4.
	319	Illegal level number for data-name reference.		602	4C element, no PNATBL entry.
	337	DEBUG-ITEM not in dictionary. UFD verb does not follow UFD section PN definition.		603	4C element, PN number less than table PN number.
35	339	UFD verb does not follow UFD section PN definition.		604	4C element, PN number greater than table PN number.
	340	No UFD verbs in declaratives.		605	50 element, GN number less than GNATBL GN number.
	341	UFD operands not followed by BCD literal.		606	50 element, GN number not in GNATBL.
40	400	Zero length items found in P1-text or source error in arithmetic, while processing SETTBL.		608	44 element, code is less than or equal to 68 and invalid.
45	450	Unexpected verbs, unrecognizable input or missing subscript string.		609	GN number not found in QGNTBL.
50	501	Error allocating even-odd pair of registers.		613	B02C found.
	502	Invalid subscript.		614	4400 found.
	506	Invalid data-type for sending field.		615	88xx or 8Cxx found, xx = 00 or 06.
	507	Invalid operand 1 or 2 for arithmetic verb.		616	8809 or 8C09 found.
	508	Invalid floating point conversion.		617	6438 found.
	509	Invalid subscript combination.		618	644C found.
	512	No match for IR in table.	65	619	8418 found.
60	605	Error found processing input.	70	621	8420 found.
62	062	Error found processing input.		622	8424 found.
				623	8448 found.
				624	8450 found.
				625	8454 found.
				626	8800 or 8C00 found.
				627	8806 or 8C06 found.
				628	78IDDDKK, found. KK ≠ 00, 01, 04, or 06.
				643	Error found processing deftext.
				065	No SEGINDX table entries.
				070	Error processing E-text.

Message	Explanation
IKF0020I	Due to a logic or machine error, a TAMER routine cannot satisfy a table handling request.
IKF19I	Maximum card number exceeded. NUM option canceled. Issued by phase 02, 10, 12, 1B.
IKF192	SYMDMP or TEST option canceled due to NUM sequence error. Issued by phase 02, 10, 12, 1B.
IKF1009	BASIS library not available.
IKF1010	Unused end of data of SYSIN during phase 04 processing.
IKF1090	Library load error during BASIS processing. No member or bad block.
IKF1114	No library member or BASIS card.
IKF1198	Insufficient compiler workspace for COPY-BASIS processing.
IKF2074I	Illegal minor code field for a RENAMES entry.
IKF2152I	Unrecognizable input to phase 20, 22, or 21. Skipped to next phase.
IKF2256	Illegal entry in UPSI table.
IKF3012I	No matched DCB in the QFILE table for a file.
IKF3013I	Pointer to dictionary entry less than the QVAR table entry for an elementary item.
IKF3014I	No match found in the QVAR table for an elementary item.
IKF3016I	Error in processing a CORRESPONDING option.
IKF3017I	Invalid minor code field in dictionary entry.
IKF3019I	Invalid level found while processing glossary.
IKF3020I	Report name is invalid as used. Discarded.
IKF3037	DEBUG-ITEM not in dictionary when expected.
IKF3039	Unexpected verb follows USE FOR DEBUGGING Section Paragraph name.
IKF3040	NO USE FOR DEBUGGING declaratives.
IKF3041	Unexpected input follows USE FOR DEBUGGING operand.
IKF4068I	Undefined data attributes.
IKF4088I	Procedure IC-text count field is 0. Skipped to phase 50.
IKF4142I	Unrecognizable input on SYSUT2.
IKF4143I	Logic error. A subscript in the SSCIN string has been lost.
IKF5001I	Logic or machine error while trying to assign a double register.

Figure 70 (Part 1 of 2). Error Messages Indicating Compiler Error

Message	Explanation
IKF5002I	Logic or machine error while processing a subscripted or indexed data-name.
IKF5005I	Logic or machine error while processing a MOVE statement.
IKF5006I	Logic or machine error. Unexpected input to the MOVE or STORE processor.
IKF5007I	Logic or machine error. Unexpected input to the arithmetic code generator.
IKF5008I	Logic or machine error. Unexpected input to the floating-point arithmetic routine FPCVBH.
IKF5009I	Logic or machine error. Lost subscript or index identifier in the XSSNT table.
IKF5012I	Logic or machine error. Lost intermediate result attributes in the XINTR table.
IKF6003I	Unknown Data A-text code found while processing on SYSUT4.
IKF6005I	Error found while processing Procedure A-text on SYSUT1.
IKF6008I	STATE option canceled.
IKF6010I	All debugging options canceled.

Figure 70 (Part 2 of 2). Error Messages Indicating Compiler Error

Module Name	Hexadecimal Displacement from Beginning of Module
IKFCBL00	5B
IKFCBL01	30
IKFCBL02	08
IKFCBL03	08
IKFCBL04	0C
IKFCBL05	08
IKFCBL06	08
IKFCBL08	08
IKFCBL10	08
IKFCBL12	08
IKFCBL1B	08
IKFCBL20	08
IKFCBL21	08
IKFCBL22	08
IKFCBL25	08
IKFCBL30	08
IKFCBL35	0C
IKFCBL40	08
IKFCBL45	08
IKFCBL50	08
IKFCBL51	08
IKFCBL60	08
IKFCBL62	08
IKFCBL63	08
IKFCBL64	04
IKFCBL65	08
IKFCBL6A	08
IKFCBL70	08
IKFCBL80	08

Figure 71. Location of Identifier Constant

#### Identifying the Version of the Compiler

A unique identifier constant (C'VSCBL200' for this compiler) is located at a fixed displacement to enable the user to determine from a storage dump which version of the compiler was used for the compilation. The only exception is module IKFCBL00 which contains copyright information instead of the identifier constant at the indicated displacement. The following information is found at hexadecimal 00 00 00: 5740-CB1 COPYRIGHT IBM CORPORATION 1974, 200143. The location of the identifier constant in the compiler is as given in Figure 71.

#### Current Phase and Record

If the compiler terminates execution while a processing phase (04, 05, 06, 08, 10, 12, 1B, 20, 22, 21, 25 3, 35, 4, 45, 50, 51, 6, 62, 63, 64, 65, 6A, 70 or 80) is in storage, the phase can be identified in three ways. In a dump the first active Request Block (RB) is for phase 00, and the second is for the processing phase. The load point for the processing phase can be found by adding hexadecimal 20 to the origin point specified in the RB; the RB contains the address of a contents directory entry (CDE) which gives the load module name and the entry point.

The name of the phase currently processing can also be found in either LINKCNT or LINKNAME, both locations in phase 00. In LINKCNT the processing phase is expressed in a 2-byte code as follows:

<u>Code</u> <u>(Hexadecimal)</u>	<u>Phase</u>
2	01
2	02
4	04
6	05
8	06
A	08
C	10
E	12
10	1B
12	20
14	22
16	21
18	25
1A	3
1C	35
1E	4
20	45
22	50
24	51
26	6
28	62
2A	63
2C	64
2E	65
30	6A
32	70
32	80

OR  
Execution of phases 70,  
71, and 72 is not  
required.

OR  
Abnormal termination  
occurred.

LINKNAME contains an 8-byte name in the form IKFCBLxx, where xx is the phase number.

The record that is being processed at the time the error occurred can be related to a statement in the listing through a location in COMMON, CURCRD, which contains the current generated card number. If there is no listing available, the buffers for the files being read or written can be located and the contents of the last bytes used can be examined. This process is described under "Buffers and Their Contents" in this chapter.

### Saving Registers

Phase 00 places the address of the control program's save area in location MYSAVE+4. When phase 00 is called by another compiler phase, it places the address of the calling phase save area in SAVER13. Phase 00 puts the address of its own save area (MYSAVE) in register 13.

The calling phase registers can be located by adding decimal 12 to the address contained in SAVER13. This locates register 14, followed by register 15, etc.

The following registers have significance in connection with a branch to COS, the location in phase 00 used for input/output requests:

- Register 0: Contains the address of the X and Y parameters of the linkage request. (See Table 2 in the chapter "Phase 00.") Register 0 can be used to verify that the calling phase is making a legitimate request, for example, phase 20 cannot write on SYSPRINT. For PLS phases (04 and 35), register 1 contains the address of the X and Y parameters.
- Register 2: Contains the address within the calling phase from which phase 00 is to write if the X parameter in register 0 is a request for a PUT. It can be used to determine whether the address it contains is within the range of main storage allocated to the calling phase.
- Register 3: Should contain the length of the data to be written if a PUT has been requested. The number must be less than the size of the buffer but not equal to zero.

If the calling phase is asking for action by a TAMER routine, the registers of the calling phase are saved in an area starting at location TBSAV1 by means of a STM 0,15 instruction. The registers can be found at A(TBSAV1). Register 14 contains the address of the instruction following the call to the TAMER routine; register 15 contains the address of the TAMER routine entry point.

### Buffers and Their Contents

When a processing phase requests an input/output operation by phase 00, register 10 in phase 00 contains the data set number code (hex 01 through 0B) multiplied by 2. This code is the

displacement of an entry for the data set within the phase 00 POINT table.

Figure 72 shows the format of a POINT table entry. Each consists of two bytes, the low-order bits of which contain buffer numbers (n). If a data set is double-buffered, the numbers are different. If it is single-buffered, the numbers are the same. If the first four bits of the entry contain a hexadecimal F, no physical input/output has been done on the data set in this phase.

Bits	Contents
0-3	X'0' or X'F'
4-7	buffer number n
8-11	X'0'
12-15	buffer number

Figure 72. POINT Table Entry Format

The buffer number can be used to locate the buffer control block (BCB) for the buffer being used for the data set. Buffer control blocks are contained in an area starting at BUFCNLS-8 (displacements are in hexadecimal) as follows:

<u>Location</u>	<u>Control Block</u>
BUFCNLS-8	Second SYSLIN BCB.
BUFCNLS+0	Buffer 1 BCB.
BUFCNLS+8	Buffer 2 BCB.
BUFCNLS+10	Buffer 3 BCB.
BUFCNLS+18	Buffer 4 BCB.
BUFCNLS+20	Buffer 5 BCB.
BUFCNLS+28	BCB for buffer 6 as a whole.
BUFCNLS+30	First SYSPRINT BCB; bytes 2 through 4 point to the beginning of buffer 6.
BUFCNLS+38	Second SYSPRINT BCB.
BUFCNLS+40	Two SYSTEMR BCBs.
BUFCNLS+50	Two SYSIN BCBs.
BUFCNLS+60	Two SYSLIB BCBs.
BUFCNLS+70	First SYSPUNCH BCB; bytes 2 through 4 point to the end of the SYSTEMR buffer area. SYSPUNCH and SYSLIN reuse the SYSIN and SYSLIB area.

BUFCNLS+78 Second SYSPUNCH BCB.

BUFCNLS+80 First SYSLIN BCB.

The format of a BCB is as follows:

<u>Bytes</u>	<u>Contents</u>
0	X'00'
1-3	Address of buffer
4-5	Bytes used so far for GET or PUT
6-7	Length of buffer

**Note:** The above does not apply to SYSUT5. Phases 25 and 65 perform buffer management processing for SYSUT5.

### Locating Tables

Tables currently being used by a phase can be located by taking the following steps:

1. The Table Information Block (TIB) number for any table handled by the TAMER routines may be found in Figure 59 (listed by phase), in Figure 60 (listed by TIB number), or under "Compiler Table Formats" (listed alphabetically by table name).
2. Add hexadecimal 34 to the load address for phase 00 given in the dump.
3. Add the result to the displacement of the TIB shown in the listing for phase 00. The result is the address of the TIB (table information block).
4. The second, third, and fourth bytes in the TIB contain the address of the TAMM for the table.
5. The second, third, and fourth bytes of the TAMM contain the address of the table. The seventh and eighth bytes contain the number of bytes used so far in the table.

The entry format for each table manipulated by TAMER routines is shown in "Section 5. Data Areas."

### DIAGNOSTIC ASSISTANCE

When you telephone the IBM Program Support Representative for diagnostic assistance, you can get a faster and more precise response if you provide as much information about the problem as possible. To determine whether your problem has been documented, it is necessary that you provide certain items, referred to as



search arguments, that are needed to retrieve such documentation. Search arguments include such items as component identification, when failure occurred, type of failure, phase that failed, and verb being processed.

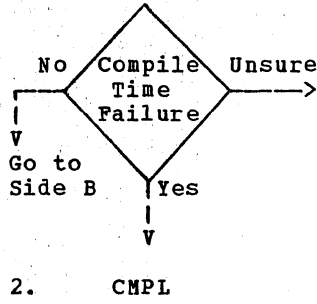
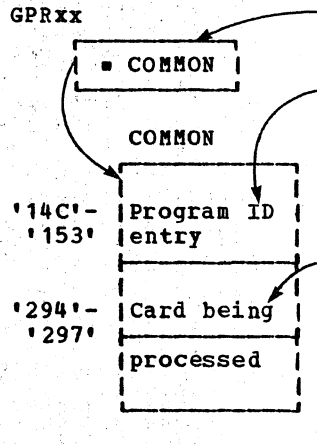
To assist you in determining search arguments, a COBOL Abstract Worksheet appears on the next page. The worksheet describes each search item, the search argument that identifies that item, and an explanation of how to find the search argument. For most efficient retrieval of

documentation regarding your problem, provide as many search arguments as you can.

In addition to the search arguments, have as many of the following items available as possible when you telephone for diagnostic assistance:

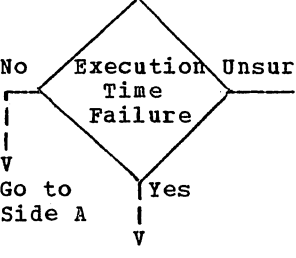
- JCL
- source listing
- dump
- console sheet
- program output

COBOL Abstract Worksheet (Side A)

Search Item	Search Argument	How to Find Argument
1. Component ID: 5740-CB1-xxx where xxx is the program product version level	1. 5740-CB1-xxx	The program product version level is printed at the top of the first page of the COBOL listing. It is also contained in byte X'14' of the load module.
2. Keyword entry (use as indicated):		If unsure, check the JCL to see which step was being executed at the time of the failure.
3. Type of failure: one of the following:  abend = ABENDxxx wait = WAIT loop = LOOP msg. = MSGIKFxxxxx	3.	
4. Failing phase: IKFCBLnn, where nn is the compiler phase number.	4. IKFCBLnn	Location LINKNAME in phase 00 contains a name in the form IKFCBLxx, where xx is the compiler phase number.
5. COBOL verb being processed: (e.g., ADD, MOVE, GO)	5.  GPRxx 	A. Find the register used by the failing phase (argument 4 above) to point to COMMON; see "Register Usage" in this section.  B. Obtain the address of COMMON from the register.  C. Verify that you are at the correct area for COMMON; the contents of COMMON address + X'14C' should be the same as the word coded on the PROGRAM-ID card (card 2 or 3 in the source program).  D. Find card number at COMMON address + X'294' (e.g., 8000030 would indicate card 48 (hex. 30 = dec. 48)).  E. Find the indicated card in the source program and determine the COBOL verb being processed.

<sup>1</sup>If this worksheet has been removed from the OS/VS COBOL Program Logic Manual, the address of COMMON can be found by using all of the registers listed under REG AT ENTRY TO ABEND. Starting with register 10 (then 11, then 9) perform steps B and C until step C results in a match; then do steps D and E.

COBOL Abstract Worksheet (Side B)

Search Item	Search Argument	How to Find Argument
1. Component ID: 5740-CB1-xxx, where xxx is the program product version level.	1. 5740-CB1-xxx	The program product version level is printed at the top of the first page of the COBOL listing. It is also contained in byte X'17' of the load module.
2. Keyword entry (use as indicated):	 <p data-bbox="592 787 738 808">2. EXEC</p>	If unsure, check the JCL to see which step was being executed at the time of the failure
3. Type of failure: one of the following: abend = ABENDxxx wait = WAIT loop = LOOP msg. = MSGIKFxxxxx bad output = INCORROUT	3.	
4. COBOL verb or statement being executed:	4.	<p data-bbox="928 1077 1507 1203">A. Determines the CSECT name assigned to the failing program in the PROGRAM-ID source statement, which is usually the second or third card in the source program.</p> <p data-bbox="928 1224 1507 1297">B. Determine the length of the compiled CSECT from one of the following areas:</p> <ul style="list-style-type: none"> <li data-bbox="1003 1325 1300 1350">• linkage editor map</li> <li data-bbox="1003 1371 1198 1396">• extent list</li> <li data-bbox="1003 1417 1463 1442">• compiler generated Memory Map</li> </ul> <p data-bbox="928 1463 1507 1537">C. Use the PSW AT ENTRY TO ABEND to determine the address of the failing instruction.</p> <p data-bbox="928 1558 1507 1663">D. Is the failing instruction within the scope of the code compiled for the COBOL CSECT? If yes, see note 1 below. If no, see note 2 below.</p>
5. Optional COBOL library module name being executed:	5.	
6. Special COBOL features being used (optional): (e.g., SORT, SYMDMP)	6.	

APPENDIX A: TABLE AND DICTIONARY HANDLING (TAMER)

The tables that the compiler uses to store information within a phase or between phases are manipulated by a set of routines called TAMER (Table Area Management Executive Routines). TAMER is part of phase 00 and is resident in storage throughout compilation.

The dictionary, which is an internal data set used by phases 1B, 22, 21, 25, and 3, is manipulated by a set of routines called ACCESS routines. ACCESS routines are loaded into storage as part of these phases.

The chapter "Phase 02" explains how an area for tables and the dictionary is obtained initially. TAMER obtains additional areas if they are needed. The new areas may or may not be contiguous to the original area. Figure 73 shows the arrangement of tables and the dictionary in each contiguous area.

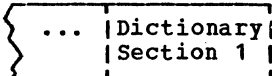
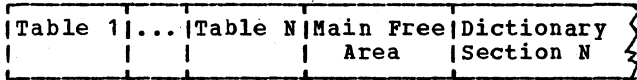


Figure 73. Arrangement of Tables and Dictionary Sections in Contiguous Areas

ACCESS DICTIONARY HANDLING ROUTINES

ACCESS routines enter and retrieve dictionary entries. They are assembled by means of a macro instruction with phases 1B, 22, 21, 25, and 3, the only phases that use the dictionary. Only those routines that are needed in a phase are assembled with it.

Using the ACCESS routines, phases 1B, 22, and 21 make dictionary entries for data-names and procedure-names in the order in which the names are defined in the source program. (Phase 1B makes entries for procedure-names; phases 22 and 21 makes entries for data-names.) Basically, a dictionary entry consists of a name and attributes.

Formats for the types of dictionary entries are illustrated in "Section 5.

Data Areas." Phases 1B, 22, and 21 do not use the LOCNXT ACCESS routine.

Phase 25 uses the ACCESS routines to locate dictionary information used in building tables for the Debug data set.

Phase 3 uses the ACCESS routines to replace data-names and procedure-names in procedure statements with their dictionary attributes. It tells the ACCESS routines the name and the ACCESS routines obtain the attributes from the dictionary entry for that name.

It also uses the ACCESS routines to resolve qualification and the CORRESPONDING option. Phase 3 does not use "enter" ACCESS routines (those whose names begin with "ENT") or the GETPTR ACCESS routine.

The ACCESS routines do not restore registers 0 and 1 on exit.

ORGANIZATION OF THE DICTIONARY

The dictionary is divided into sections of 512 bytes each. The location of a dictionary entry is indicated by its section and its displacement from the beginning of that section. The DICOT table has an entry for each dictionary section, giving the starting address of that section.

The ACCESS routines use the HASH table to keep track of the locations of dictionary entries. HASH is a hash table with 521 entries. When a dictionary entry is made for a name, its section and displacement are entered in the HASH table entry for that name. When an ACCESS routine wants to find an entry, it determines the hash value of the name to find the HASH table entry for that name and obtains the section and displacement from the HASH table entry.

If a name hashes to the same value as a previous name, the section and displacement for the previous name are taken from table HASH and placed in front of the dictionary entry for the new name as a dictionary pointer. Then the section and displacement of the new entry are entered in the HASH table and duplicate hash values are indicated. When an ACCESS routine wants to find an entry for a name and the HASH table indicates that there were duplicate hash

values, it uses these dictionary pointers to search the dictionary, in reverse order, to find the specified name. That is, it obtains the section and displacement from the HASH table for the hash value and finds the name in the indicated entry. If the names match, this is the correct entry, unless duplicate names were defined. Then it looks at the entry with the section and displacement specified by the dictionary pointer of the last entry. This process continues until it has compared the names of all entries with duplicate hash values. At that time, it will either have found a single unique name, a duplicately defined name, or no name at all. It issues an error code if it does not find a name or if the name is duplicately defined.

**Note:** If a name is unique because it is qualified, the phases specify a range of dictionary entries to be searched when they call an ACCESS routine. This is explained in routine LATACP.

**STORAGE FOR THE DICTIONARY**

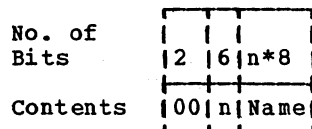
Generally, storage for a new dictionary section can be obtained from the free area between the tables and the dictionary (see Figure 73). If the free area is exhausted, a GETMAIN macro instruction is issued to obtain all available COBOL space, the area equal to the difference between the length of the longest phase and the length of the current phase. Areas obtained in this manner are not contiguous.

When the outstanding available area is exhausted, dictionary sections are written out on direct-access data set SYSUT1 and the area is reused. The DICOT table is used to keep track of the dictionary sections. There is an entry for each section, giving the beginning address of the section and an indication of whether or not it has been written out (spilled). A dictionary section is read back into storage, when it is needed, by subroutine MOVDIC, described under "Table Handling with TAMER" in this appendix. At the end of phase 3, all dictionary space in storage and the DICOT table and the HASH table are released. SYSUT1, the dictionary spill data set, is closed so that it may be reopened as a utility data set for later phases.

When control passes from a smaller to a larger phase, any tables or dictionary sections in danger of being overlaid by the new phase must be moved to upper storage. This possibility exists when space for the table or dictionary section was obtained via a GETMAIN macro instruction, as

described above. The necessary moving is done during interphase processing by routines named TBINTP00, TBINTP01, etc.

In the following descriptions of the ACCESS routines, the format of the EBCDIC name pointed to is:



where n is the number of characters in the name. The entry starts on a fullword boundary and is a multiple of 4 bytes. Padding is with binary zeros starting with the low-order bytes.

**INITIALIZATION OF ACCESS ROUTINES**

To use the ACCESS routines, the phases must call routine INTACC to initialize them. INTACC primes the DICOT table upon receiving control for the first time and performs other initialization functions. The call to INTACC must follow the call to TAMEIN, the TAMER initialization routine. The calling sequence to INTACC is:

```
L      15,=A (INTACC)
BALR   14,15
```

**ACCESS ROUTINES**

ACCESS routines are available to perform the following functions.

1. Enter attributes when given a data-name.
2. Enter attributes when given the dictionary pointer.
3. Get a dictionary pointer when given the data-name and the length of its attributes.
4. Enter delimiter (dictionary pointer of the entry that delimits a group or section) when given the dictionary pointer of the group or section.
5. Locate attributes when given a data-name.
6. Locate attributes when given a dictionary pointer.

7. Locate attributes of next entry when given a dictionary pointer.
8. Locate delimiter when given a data-name.
9. Locate attributes when given ACCESS pointer (name of an entry that is a subfield of the last entry referred to by an ACCESS routine).
10. Locate attributes when given a data-name and a dictionary pointer to a group of which it is a subfield.

where parameter has the following format starting on a fullword boundary:

No. of Bytes	1	3	1
Contents	16	Address of EBCDIC Name	Count of Attributes

	3	1	3
	Address of Attributes	0	Dictionary Pointer

ENTNAM (Enter Attributes Given Name)

Given the address of an EBCDIC name (procedure-name or data-name), routine ENTNAM makes a dictionary entry for it. It places its section and displacement in the HASH table and also in register 1, to be used by the calling phase as a dictionary pointer. The calling sequence is:

```
L      1,=A(parameter)
L      15,=A(ENTNAM)
BALR   14,15
```

GETPTR (Get Pointer)

Given an EBCDIC name and the length of its attributes, routine GETPTR determines its section and displacement and places this dictionary pointer in register 1. The calling sequence is:

```
L      1,=A(parameter)
L      15,=A(GETPTR)
BALR   14,15
```

where parameter has the following format starting on a fullword boundary:

No. of Bytes	1	3	1	3
Contents	Code	Address of EBCDIC Name	Count of Attributes	Address of Attributes

where parameter has the following format starting on a fullword boundary:

No. of Bytes	1	3	1
Contents	Code	Address of EBCDIC name	Count of Attributes

where code is 0 for elementary items and paragraph-names, and 4 for group items and section-names.

where code is 8 for elementary items and paragraph-names, and 12 for group items and section-names.

ENTPTR (Enter Attributes Given Pointer)

Given a dictionary pointer, that is, a section and displacement, routine ENTPTR enters a specified name and its attributes into the dictionary. The calling sequence is:

```
L      1,=A(parameter)
L      15,=A(ENTPTR)
BALR   14,15
```

ENTDEL (Enter Delimiter Pointer)

Given a dictionary pointer for a group item or section-name and its delimiter pointer, routine ENTDEL enters the delimiter pointer into the dictionary entry for the group item or section-name. A delimiter pointer for a group item is the section and displacement of the next group item on the same or a lower level. A delimiter pointer for a section-name is the section and displacement of the next section-name. The calling sequence is:

```
L      1,=A(parameter)
L      15,=A(ENTDEL)
BALR   14,15
```

where parameter has the following format starting on a fullword boundary:

No. of Bytes	1	3	1	3
Contents	0	Dictionary Pointer	0	Delimiter Pointer

LATRNM (Locate Attributes Given Name)

Given an EBCDIC name, routine LATRNM locates its dictionary pointer and the starting address of its attributes. If the entry is found, the attributes starting address is placed in register 2, the dictionary pointer is placed in register 3, and register 15 is set to 0. If the entry is not found, the contents of registers 2 and 3 are meaningless, and register 15 contains a 4 if the name was not found and an 8 if the name was duplicately defined. The address of the located name as it appears in the dictionary will be saved in the DICTNAME in COMMON. Note: This address should never be saved across CALLS to phase 00. The calling sequence is:

```
L 1,=A(parameter)
L 15,=A(LATRNM)
BALR 14,15
```

where parameter has the following format starting on a fullword boundary:

No. of Bytes	1	3
Contents	0	Address of EBCDIC Name

LATRPT (Locate Attributes Given Pointer)

Given a dictionary pointer for an entry, routine LATRPT locates the starting address of its attributes and places it in register 2. The address of the located name as it appears in the dictionary will be saved in the DICTNAME in COMMON. Note: This address should never be saved across CALLS to phase 00. The calling sequence is:

```
L 1,=A(parameter)
L 15,=A(LATRPT)
BALR 14,15
```

where parameter has the following format starting on a fullword boundary:

No. of Bytes	1	3
Contents	4	Dictionary Pointer

LOCNXT (Locate Next Entry)

Given the dictionary pointer of an entry, routine LOCNXT locates the next entry. In register 2, it places the starting address of the next entry's attributes. In register 1, it places the dictionary pointer of the next entry. In register 3, it places the starting address of the EBCDIC name of the next entry. The calling sequence is:

```
L 1,parameter
L 15,=A(LOCNXT)
BALR 14,15
```

where parameter has the following format starting on a fullword boundary:

No. of Bytes	1	3
Contents	0	Dictionary Pointer

LDELNM (Locate Delimiter Given Name)

Given the EBCDIC name of a group item or a section, LDELNM locates its delimiter. It places the starting address of the given name's attributes in register 2, the dictionary pointer of the data-name in register 3, and the delimiter pointer in register 1. It sets register 15 to 0.

If an error is detected, one of the following codes is placed in register 15. The address of the located name as it appears in the dictionary will be saved in the DICTNAME in COMMON. Note: This address should never be saved across CALLS to phase 00.

Code in Register 15	Meaning
12	Unique name located was paragraph name or elementary item name. Registers 2 and 3 are set as above.
8	Name is duplicately defined. Registers 2 and 3 contain meaningless information.
4	Name was not found. Registers 2 and 3 contain meaningless information.

The calling sequence for LDELNM is:

```
L    1,=A(parameter)
L    15,=A(LDELNM)
BALR 14,15
```

where parameter has the following format starting on a fullword boundary:

No. of Bytes	1	3
Contents	16	Address of EBCDIC Name

LATACP (Locate Attributes Using ACCESS Pointer)

Given the EBCDIC name of an entry that is a subfield of the last entry referred to by an ACCESS routine, routine LATACP puts the starting address of the attributes in register 2 and the dictionary pointer in register 3. The address of the located name as it appears in the dictionary will be saved in the DICTNAME in COMMON. **Note:** This address should never be saved across CALLS to phase 00. Register 15 is set to 0. (This routine is used to locate qualified names. It limits the search of the dictionary.)

If an error is detected, one of the following codes is placed in register 15.

Code in Register 15	Meaning
12	Last entry referred to was an elementary item.
8	Name is dublicately defined.
4	Name was not found.

Registers 2 and 3 contain meaningless information.

The calling sequence for LATACP is:

```
L    1,=A(parameter)
L    15,=A(LATACP)
BALR 14,15
```

where parameter has the following format starting on a fullword boundary:

No. of Bytes	1	3
Contents	8	Address of EBCDIC Name

**Note:** There must have been no call to a TAMER routine intervening between this routine and the last call to an ACCESS routine.

LATGRP (Locate Attributes Given Group Pointer)

Given the EBCDIC name of an entry and the dictionary pointer of the group item or section-name of which it is a subfield, routine LATGRP puts the starting address of the entry's attributes in register 2 and the dictionary pointer of the entry in register 3. It sets register 15 to 0. The address of the located name as it appears in the dictionary will be saved in the DICTNAME in COMMON. **Note:** This address should never be saved across CALLS to phase 00.

If an error occurs, one of the following codes is placed in register 15.

Code in Register 15	Meaning
12	Given dictionary pointer pointed to a paragraph-name or elementary item name.
8	Name is dublicately defined.
4	Name was not found.

Registers 2 and 3 contain meaningless information.

The calling sequence for LATGRP is:

```
L    1,=A(parameter)
L    15,=A(LATGRP)
BALR 14,15
```

where parameter has the following format starting on a fullword boundary:

No. of Bytes	1	3	1	3
Contents	12	Address of EBCDIC Name	0	Group or Section Dictionary Pointer

The address of the located name as it appears in the dictionary will be saved in the DICTNAME in COMMON. **Note:** This address should never be saved across CALLS to phase 00.

TABLE HANDLING WITH TAMER

TAMER (Table Area Management Executive Routines) resides permanently in main storage as part of phase 00 and is available to all phases to handle tables.



**CONTROL FIELDS**

The three control fields described below are set up and used by the TAMER routines as aids in the handling of tables.

TIBs (Table Information Blocks)

For each table, there is a TIB in a fixed location in COMMON. A TIB may be reassigned when the table for which it was used is released. The TIB points to another control field for that table -- the TAMM (see below). Each TIB has the following format:

No. of Bytes	1	3	2	2
Contents	Entry Length	TAMM Address	Table Length	Growth Factor

**Entry Length**  
number of bytes in a table entry.

**TAMM Address**  
address of the TAMM for the table.

**Table Length**  
number of bytes requested for the table (used by the PRIME routine).

**Growth Factor**  
not used -- a table is always increased 256 bytes at a time.

TAMMs (Table Area Management Maps)

For each table there is a TAMM in a variable location within a fixed block (the TAMM block). Each TAMM points to a table and to the TIB for the table. The format of a TAMM is:

No. of Bytes	1	3	2	2	4
Contents	Status	Table Address	N1	N2	TIB Address

**Status**  
code indicating the status of the table:

<u>Code</u>	<u>Meaning</u>
01	Indicates that the table has been released so that its area is available as free area.
02	Indicates that the table is static. No further entries will be made.
04	Indicates that the table has been primed so that entries can be made.

**Table Address**  
address of the first byte of the table.

N1:

<u>If Status Is</u>	<u>N1 Is</u>
01	0
02 or 04	Number of bytes used so far

N2:

<u>If Status Is</u>	<u>N2 Is</u>
01	Length of the freed area
02	Number of unused bytes in the table
04	Number of bytes assigned to the table

**TIB Address**  
address of the TIB for the table.

MASTAMs (Master TAMM Tables)

Each MASTAM contains the characteristics of a TAMER area (contiguous space in storage assigned to TAMER). A MASTAM has the following format:

← fullword →
Beginning of area
Length of area
First free byte not used so far
Length of free area left over
First TAMM used by this MASTAM
Next TAMM to be used by this MASTAM
Number of dictionary section within the MASTAM

Each time a new TAMER area is assigned to TAMER through a GETMAIN macro instruction, one of two conditions can be true:

1. The new TAMER area is contiguous to a current one, in which case an old MASTAM will be updated.
- or
2. It is not contiguous to a current TAMER area, in which case a new MASTAM is created.

Since only three MASTAMS, at most, are needed (see "How Space is Assigned" below), provisions are made for using only three consecutive GETMAINS without an intervening FREEMAIN macro instruction.

#### HOW SPACE IS ASSIGNED

At the beginning of compilation, a variable unconditional GETMAIN is issued to get the main TAMER area. A MASTAM is set up for the area. If more space is needed later on, COBOL space is requested through an unconditional GETMAIN [COBOL space is the difference in length between the longest phase (phase 4) and the current phase]. If the COBOL space obtained is contiguous to an old TAMER area, it is considered part of that TAMER area and the old MASTAM is then updated.

If the next phase to process is not larger than the current phase, the COBOL space is kept. Since this space is kept only three MASTAMS are required. If the next phase to process is larger than the current phase, the tables that are to be passed between phases are packed and moved so that none of them are overlaid by the next phase.

Within a TAMER area (and as indicated in the MASTAM), the tables start in lowest storage and the dictionary starts in highest storage. The TAMMs for the MASTAM are assigned contiguously within the TAMM block, and their order reflects the storage order of the tables to which they point.

**Note:** If several GETMAINS are issued, the areas obtained will usually be in low storage so that only one MASTAM will be used.

#### TAMEIN Routine

TAMEIN is the TAMER initialization routine. It is called during phase 00 interphase processing (in routine INT1B) and once

before phase 3 processing. It is also called by phase 02. Its operations are as follows:

#### • Phase 02:

Called before any other TAMER routine. Sets up the first MASTAM for the TAMER area requested at the beginning of compilation. Sets up a TAMM and TIB for the HASH table (see "ACCESS Dictionary Handling Routines" at the beginning of this appendix).

#### • Phase 3:

If the dictionary was not spilled (that is, if no dictionary sections were written on SYSUT1), no action is taken. If the dictionary is spilled, TAMEIN calls TBGETSPC which tries to get more space for dictionary sections. If space is available, TBREADIC is called to read back as many sections as possible.

The calling sequence for TAMEIN is:

```
L      15,=A(TAMEIN)
BALR   14,15
```

#### PRIME Routine

Routine PRIME allocates space, in any TAMER area, to the table named in the calling sequence. The following steps are taken in sequence until the required space is found:

1. A check is made for the required area in the remaining free space of the current TAMER area. This space is called the Main Free Area. If the area is too small, the Main Free Area contained in the other TAMER areas is checked.
2. A check is made for the required area in the space made available by the release of tables by TAMER. This space is called the Freed Area.
3. An attempt is made to pack the tables (eliminating the free bytes between the primed and static tables) to increase the size of the Main Free Area within a TAMER area.
4. A request is made for COBOL space.
5. The primed tables are packed, that is, all of the unused area minus the length of one more entry for each table is considered available.
6. An attempt is made to spill a dictionary section (write it out on

SYSUT1). For steps 5 and 6, table space is assigned only on a single entry basis.

If none of these methods is successful, compilation cannot continue. If space is found, a TAMM is created for the new table (or is just updated in the case of success in step 2).

The PRIME routine can also be called internally by other routines just to find space. In this case, a TAMM is not created and the calling routine takes whatever action is necessary.

The calling sequence for PRIME is:

```
L      1,=A(parameter)
L      15,=A(PRIME)
BALR  14,15
```

where parameter has the following format starting on a fullword boundary:

No. of Bytes	1	3	2	2
Contents	Entry Length	TIB Address	Requested Size	Growth Factor

TBGETSPC Routine

Routine TBGETSPC is called by routine PRIME to obtain COBOL space. The space is requested through an unconditional GETMAIN macro instruction. A check is made to determine whether or not the space obtained is contiguous to a current TAMER area. If the space is contiguous and is in lower storage than the TAMER area, all of the tables in the TAMER area are moved to what is now the new beginning of the TAMER area. If the space is not contiguous, a new MASTAM is created, and the COBOL space becomes a new TAMER area.

MOVDIC Routine

Routine MOVDIC reads back into storage a dictionary section that has been spilled. First, MOVDIC calls the PRIME routine to make space available for the section and then it reads the section back into that space.

The calling sequence for MOVDIC is:

```
L      15,=A(MOVDIC)
L      3,=A(DICOTABLE ENTRY)
BALR  14,15
```

DICSPC Routine

Routine DICSPC is called only by ACCESS routines, and requests space for a dictionary section. The space is provided by an internal call to the PRIME routine. PRIME returns the starting address of the section in register 1, and the ending address in register 2.

The calling sequence for DICSPC is:

```
L      15,=A(DICSPC)
BALR  14,15
```

STATIC Routine

Routine STATIC renders a table static. This means that no new entries will be made in the table during the rest of the phase. It sets the TAMM for the table to static format, that is, to the form:

No. of Bytes	1	3	2	2
Contents	02	Table Address	Used Bytes	Free Bytes

The calling sequence to routine STATIC is:

```
L      1,=A(TIB)
L      15,=A(STATIC)
BALR  14,15
```

TABREL Routine

Routine TABREL releases a table when it is no longer needed so that its area can be used as a free area. It sets the TAMM for the table to released format, as follows:

No. of Bytes	1	3	4
Contents	01	Table Address	Released Bytes

The TAMM address in the TIB is set to 0. Both the TAMM and the TIB for the released table can now be used for another table in a call to routine PRIME.

The calling sequence for routine TABREL is:

```
L      1,=A(TIB)
L      15,=A(TABREL)
BALR   14,15
```

#### INSERT Routine

Routine INSERT provides for inserting an entry into a table. It adjusts the displacement field of the TAMB for the table and returns to the phase the starting address (in register 2) and the starting displacement (in register 3) of the entry.

If the area allocated to the table will not hold the entry, routine INSERT calls PRIME to obtain additional space.

The phases call INSERT with the following calling sequence:

```
L      1,=A(TIB)
L      15,=A(INSERT)
BALR   14,15
```

Note: If a table contains variable-length entries, the entry length specified in the TIB must be changed before a phase calls routine INSERT.

#### TAMEOP Routine

Routine TAMEOP is called in the interlude before every phase to reset TAMER switches and to ensure that no tables being passed from a shorter phase are overlaid by a longer phase. In phase 3, it releases dictionary space.

The calling sequence for TAMEOP is:

```
L      15,=A(TAMEOP)
BALR   14,15
```

#### TBSPILL Routine

Routine TBSPILL, which is called by PRIME, checks for the last dictionary section (the section in highest storage), and calls routine TBWRITE to spill it to provide additional storage for tables. If the section in highest storage is currently being built, the next-to-last one will be spilled.

After the dictionary section is spilled, TBSPILL moves the first section (the section in lowest storage, and, therefore, closest to the tables) to the area just freed.

#### TBWRITE Routine

Routine TBWRITE is called by routine TBSPILL to spill dictionary sections by writing them on the direct-access data set SYSUT1. TBWRITE uses the DICOT table to check and indicate the status of dictionary sections (see "ACCESS Dictionary Handling Routines" at the beginning of this appendix).

If a section has never been spilled, TBWRITE spills it by issuing the BSAM WRITE macro instruction. If a section has been spilled before, but has been changed since then, TBWRITE issues the XDAP macro instruction to put the fresh copy on SYSUT1. If a section has been spilled and has not been changed since then -- that is, an exact copy already exists on SYSUT1 -- it is not spilled again.

#### TBREADIC Routine

Routine TBREADIC is called by MOVDIC to read a dictionary section back into storage.

#### GETALL Routine

Routine GETALL is called by phases 6 and 6A. Its function is to provide space for a table that may be in excess of 32K bytes, the normal maximum size. It is a request for all available table space in a contiguous area. The tables are packed and, if all available COBOL space has not yet been acquired, a GETMAIN macro instruction is issued for the remainder. The largest unused, contiguous area is found, and its starting address and length are passed back to the calling phase in registers 0 and 1, respectively.

All subsequent use of that area is handled internally by the phase that called GETALL, since a call to TABREL is the only TAMER call which may legitimately follow a call to GETALL in the same phase. At the end of the phase that called GETALL, the area then becomes available for normal phase 00 table-handling procedures.

APPENDIX B: OBJECT MODULE

The compiler produces an object module suitable for input to the linkage editor. After linkage editing, the module is arranged as shown in Figure 74.

If the program is segmented, there are some differences in the object module. These are described in the section "Segmented Object Module" later in this appendix.

OVERVIEW OF OBJECT MODULE FIELDS

This section provides a brief description of the fields of the object module as they appear in Figure 74. Each field is discussed in greater detail elsewhere in this appendix.

**INIT1**  
the initialization 1 routine at relative location 0 sets up registers and address constants for the program.

**DATA AREA**  
data areas are data constants defined in the Data Division. This includes FIBs, DCBs, DECBS, record areas, and some buffers.

**TGT**  
the Task Global Table contains information and work areas used by the program.

**PGT**  
the Program Global Table contains address constants and literals referred to by procedure instructions.

**RPT**  
Report Writer routines to process the Report Section. This field is discussed in "Appendix C: Report Writer Subprogram."

**PROCEDURE**  
procedure instructions for the program. In a segmented program, this field contains only the instructions for the root segment.

**Q-ROUTINES**  
compute the length of variable-length fields defined by the OCCURS clause with the DEPENDING ON option, and the locations of the variably-located fields which follow them.

**COUNT TABLE**  
the COUNT table contains entries for each procedure-name and source verb (COUNT option).

**INIT2**  
the initialization 2 routine stores address constants as part of program initialization.

**INIT3**  
the initialization 3 routine establishes addresses in the global tables.

**PROCTAB and SEGINDX**  
the PROCTAB and the SEGINDX tables contain debugging information created in response to a request for the statement number (STATE) option.

INIT1
DATA AREA
TGT
PGT
RPT
PROCEDURE
Q-ROUTINES
COUNT TABLE
INIT2
INIT3
PROCTAB
SEGINDX

Figure 74. Storage Map of a COBOL Object Module

INITIALIZATION 1 ROUTINE (INIT1)

The initialization 1 routine begins at relative location 0. The coding of the routine is identical for every program; however, the address constants it contains differ with program requirements. These address constants were provided by phase 6

or 64. This routine performs the following functions, in the order listed:

1. Saves the registers of the calling program and the pointer to its Task Global Table or save area.
2. Establishes address constants for this program's Task Global Table, Program Global Table, first instruction to be executed, and initialization 1, 2, and 3 routines.
3. Branches to INIT2 (described later in this chapter).
4. Branches to ILBOINTO COBOL Library Subroutine to do initialization for VSAM file processing.

In addition to its initialization functions, the INIT1 routine also contains the following:

1. A DCB exit routine for the QISAM delete code.
2. A "no-space" condition exit routine for BSAM and QSAM files.
3. A DCB exit routine for a BISAM file CORE-INDEX option, if specified in the program.
4. A save area. This is used if the program is re-entered. It contains the register contents as they were left when the program was last executed, to be loaded by INIT3.

DATA AREA

The data area of the COBOL program contains space for information provided in the source program Data Division. It holds control blocks, buffers, and record areas for files, and reserved storage for the Working-Storage Section, with data items initialized if the VALUE clause was specified.

The format of the data area is shown in Figure 75.

In the first part of the data area, all the information for each file is contiguous in storage. The information exists in the order in which FDS were written in the source program.

The data area was generated by phase 22, which indicated space allocation by setting the LOCCTR cell in COMMON and which placed some constants in exit lists, DCBs, DECBS, and Working-Storage where the VALUE clause was specified.

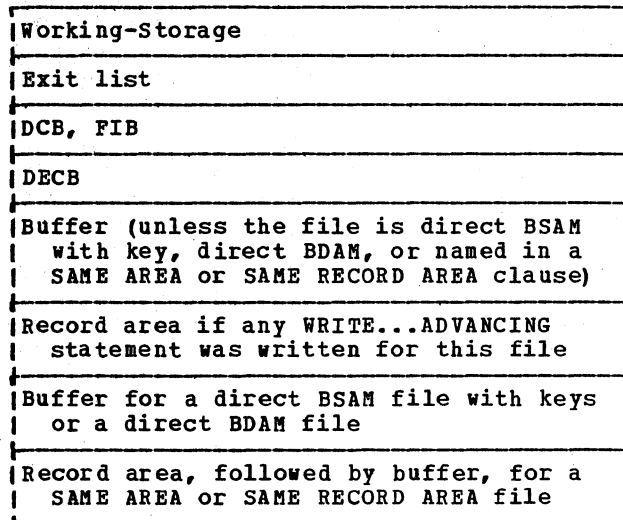


Figure 75. Format of the Data Area

Formats for DCBs (data control blocks) and DECBS (data event control blocks) may be found in OS/VS1 System Data Areas, and OS/VS System Data Areas.

A special discussion is given for exit lists since the system-defined format is not specific, and the compiler always generates exit lists with specific format and contents.

EXIT LISTS

The compiler generates an exit list for each file. At execution time, the exit list is located in storage immediately preceding the DCB for that file. Exit list entries are accessible to the COBOL program because of their location relative to the DCB. The list is accessed by the operating system via the DCBEXLST field of the DCB. This field contains the address of the exit list, which was placed there by phase 21.

General information on exit lists may be found in the publication OS/VS Data Management Services Guide.

The compiler always generates the exit list in the order specified in Figure 76. If the object program does not contain one of the routines specified, the exit list entry is present, but deactivated.

The primary function of the exit list is to provide the addresses of routines which are to be called by the operating system under certain conditions. However, the COBOL-generated exit list contains

additional information for use by the COBOL program. These exit list entries are masked off from the system by having a hexadecimal '00' in the first byte of the entry.

Figure 76 shows the format of the exit list and the text that follows describes each entry, which is a fullword in length.

- Code 1 may be:

Value	Meaning
01	input
02	output
00	inactive

- Code 2 may be:

Value	Meaning
04	output
00	inactive
0C	input SUL
0D	input NSL

**BOF, BOV-RTN**

address of a routine to process header labels. This routine was written in the source program as a USE... [BEGINNING] LABELS declarative section. The same entry is used for both the FILE (beginning-of-file) and REEL/UNIT (beginning-of-volume) options, since the system does not distinguish between beginning of file and beginning of volume. The entry is initialized before the file is opened to contain the address of the beginning-of-file routine. Immediately after the file is opened, the address of the beginning-of-volume routine is moved into the entry. The instructions which change the entry are part of the COBOL-generated code for the OPEN verb.

If the user did not provide a beginning-of-file label routine, the entry is initially inactive. If no beginning-of-volume routine was provided, the entry is deactivated after the file is opened.

**EOV-RTN**

address of the routine to process end-of-volume trailer labels. This routine was written in the source program as a USE...[ENDING][REEL/UNIT] LABELS declarative section. The entry is initialized at OPEN time.

←-----fullword-----→	
code 1	A (BOF,BOV-RTN)
code 2	A (EOV-RTN)
code 2	A (EOF-RTN)
0A	A (TOTALING AREA)
05	A (DCB-EXIT-RTN)
08	A (ERROR-BIT-RTN)
00	A (INV-KEY-GN)
00	A (USE-ERROR-RTN)
00	reserved
00	n  NSL Max. Size
06	=V (ILBOCKP1)
80	A (CKPTDCB)
.....DCB.....	

Figure 76. Fields of the Exit List

**EOF-RTN**

address of the routine to process end-of-file trailer labels. This routine was written in the source program as a USE...[ENDING][FILE] LABELS declarative section. The entry is initialized at OPEN time.

**TOTALING AREA**

address of the area specified as the TOTALING AREA data-name in the LABEL RECORDS clause.

**DCB-EXIT-RTN**

address of a routine called by Data Management during execution of OPEN for all QSAM or QISAM files and for BISAM files with the CORE-INDEX option. For QISAM, the routine sets the delete code in the DCBOPTCD field of the DCB. For BISAM, it initializes fields in the DCB for CORE-INDEX. The routine for QISAM or BISAM is located in INIT1. The exit list entry for these files is set by phase 21 for QSAM, the address is that of the COBOL library subroutine ILBOEXT1 which sets the DCBRECFM field of the DCB. The exit list entry for a QSAM file is set by phase 51.

The instructions necessary for the QISAM DCB exit routine are always generated, even if the program uses no QISAM files. Phase 6 or 64 determined whether to generate the BISAM DCB exit routine by testing the INITBIT bit of the PH1BYTE cell in COMMON.

This bit is turned on during compilation if CORE-INDEX was specified.

**ERROR-BIT-RTN**

address of a routine used for BSAM and QSAM files, if a WRITE statement failed to execute because of a no-space condition. This routine sets the return code to indicate why the WRITE statement failed, and returns to the system, which then takes the SYNAD exit. The coding for this routine is located in the INIT1 routine. It is always generated, even if no BSAM or QSAM files are used in the program. The exit list entry pointing to the routine is set by phase 21.

**INV-KEY-GN**

address of the routine to handle an INVALID KEY condition for a particular READ, WRITE, REWRITE, or START statement. If no routine was written by the user, this entry is rendered inactive by placing a 1 in the low-order bit.

The contents of this entry are changed or rendered inactive before each READ, WRITE, REWRITE, or START statement executed for the file. This is done because each INVALID KEY routine applies only to the statement for which it was written. When an input/output operation is executed, this entry always contains the address of the proper routine, or it is rendered inactive if no routine applies.

The exit list entry is masked off from the system. It is not given control directly by the system, but by COBOL-generated code in the following manner. When an error occurs during an input/output operation, the system gives control to the routine specified in the DCBSYNAD field of the DCB. This is the COBOL library subroutine ILBOSYN. If the error was caused by an invalid key, ILBOSYN checks for an active entry in this exit list position. If one is found, control is passed to the address specified, that is, the user's INVALID KEY routine. If none is found, but an active entry is found in the USE-ERROR-RTN exit list position for this file, control is passed to the USE...ERROR routine. If neither of these routines was written by the user, the ILBOSYN routine

takes its own actions, as described in the publication IBM OS/VS COBOL Subroutine Program Logic.

**USE-ERROR-RTN**

address of a routine to handle input/output errors for non-VSAM files. This routine was written by the user as a USE AFTER STANDARD ERROR declarative section. It is masked off from the system, and is given control after an input/output error in the same manner as the INVALID KEY routine, through the COBOL library subroutine ILBOSYN. This entry is interrogated first for any input/output error which is not caused by an invalid key condition. The address is placed in this entry when the file is opened. FS is posted if present.

**NSL Max. Size**

maximum number of bytes required for nonstandard labels for this file. If nonstandard labels are used, the user has written an SVC routine to process these labels, and has incorporated this routine into the system at installation time. This entry is available for interrogation by the SVC routines. It is masked off from the system's exit list examination. Further description of the requirements for nonstandard labels may be found in the publication IBM OS/VS COBOL Compiler and Library Programmer's Guide.

- n represents:

<u>Value</u>	<u>Meaning</u>
00	input
04	output
08	I-O

**ILBOCKP1**

address of a second entry point to ILBOCKP0. The checkpoint subroutine is entered through ILBOCKP1 when the "END OF REEL/UNIT" option is specified in the RERUN clause.

**CKPTDCB**

address of the checkpoint file DCB for a file in which checkpoints are being taken at "END OF REEL/UNIT."



TASK GLOBAL TABLE (TGT)

The Task Global Table is used to record and save information needed during execution of the object program. It begins with a series of fixed-length fields followed by a series of variable-length fields. These fields are illustrated in Figure 77 and are described in this section.

Relative Location		Field	Relative Location		Field
Hex	Dec		Hex	Dec	
0	0	SAVE AREA	1F4	500	PCS LIT PTR
48	72	SWITCH	1F8	504	DEBUGGING
4C	76	TALLY	1FC	508	CD for INITIAL INPUT
50	80	SORT SAVE	200	512	OVERFLOW CELLS
54	84	ENTRY-SAVE			beginning of BL CELLS
58	88	SORT CORE SIZE			Variable-length DECBADR CELLS
5C	92	RET CODE			portion FIB CELLS
5E	94	SORT RET			DEBUG TRANSFER
60	96	WORKING CELLS			DEBUG CARD
190	400	SORT FILE SIZE			DEBUG BLL
194	404	SORT MODE SIZE			DEBUG VLC
198	408	PGT-VN TABLE			DEBUG MAX
19C	412	TGT-VN TABLE			Reserved
1A0	416	Reserved			DEBUG PTR
1A4	420	LENGTH OF VN TBL			TEMP STORAGE
1A6	422	LABEL RET			TEMP STORAGE-2
1A7	423	Reserved			TEMP STORAGE-3
1A8	424	DBG R14SAVE			TEMP STORAGE-4
1AC	428	COBOL INDICATOR			BLL CELLS
1B0	432	A (INIT1)			VLC CELLS
1B4	436	DEBUG TABLE PTR			SBL CELLS
1B8	440	SUBCOM PTR			INDEX CELLS
1BC	444	SORT-MESSAGE			SUBADR cells
1C4	452	SYSOUT DDNAME			ONCTL CELLS
1C5	453	Reserved			PFMCTL CELLS
1C6	454	COBOL ID			PFMSAV CELLS
1C8	456	COMPILED POINTER			VN CELLS
1CC	460	COUNT TABLE ADDRESS			SAVE AREA=2
1D0	464	Reserved			SAVE AREA=3
1D8	472	DBG R11 SAVE			XSASW CELLS
1DC	476	COUNT CHAIN ADDRESS			XSA CELLS
1E0	480	PRBL1 CELL PTR			PARAM CELLS
1E4	484	Reserved			RPTSAV AREA
1E9	489	TA LENGTH			CHECKPT CTR
1EC	492	Reserved			VCON TBL
					DEBUG TABLE

Figure 77. Fields of the Task Global Table

The lengths of the variable-length fields are determined by the requirements of the program (if not required, a particular field may not exist in the object program). The lengths of these fields are computed in phase 6 or 62. In the chapter "Phase 6," Figure 36 shows the source of each variable TGT field.

**SAVE AREA**

the program's save area; used to provide standard subroutine linkage when this program is called (by the Operating System or by another program) and when this program calls other programs.

**SWITCH**

a fullword switch. Only the following bits are used:

Bit	Meaning
0	Indicates a size error in series addition or subtraction. If a SIZE ERROR clause was included in the source statement, and a size error occurs before all data items in the series have been added or subtracted, this bit is set to 1. It is tested after the entire addition or subtraction is complete. If the value is 1, the

instructions generated for the ON SIZE ERROR clause are executed.

- |   |  |
|---|--|
| <p>1 Used for TRACE. It is set to 1 by the execution of a READY statement, and reset to 0 by a RESET statement. If the program uses a TRACE statement, there are instructions to test this bit at the point of definition for every source program procedure-name (PN). If it is on, the DISPLAY subroutine (ILBODSP0) is called to print the card number of the procedure-name (see the publication <u>IBM OS/VS COBOL Subroutine Library, Program Logic</u> for a description of the DISPLAY subroutine).</p> <p>2 Indicates program initialization. Set to 1 by routine INIT3 to show that initialization has been performed. Tested by INIT3 so that if the module is re-entered, INIT3 can perform re-entry functions instead of initialization functions. See "Initialization 3 Routine" in this appendix.</p> <p>3 Main or subprogram switch. Set by INIT2, if this is a main program (see the subroutine ILBOSTP0 in the publication referred to above).</p> <p>4 Used for SYMDMP. It is set to 1 by phase 65 if the symbolic debug option is in effect for the program. This bit is tested by the object-time COBOL library debugging control subroutine ILBODBG0.</p> <p>5 Used for FLOW. It is set to 1 by phase 65 if the flow trace option is in effect for the program. This bit is tested by the object-time COBOL library debugging control subroutine ILBODBG0.</p> <p>6 Compile-time STATE bit. It is set to 1 by phase 65 if the statement number option is in effect for the program. At execution time, this bit is relocated to bit 10 by ILBODBG0. At execution time, this bit is set to 1 by ILBORECO when the MESSAGE condition being tested is true; it is also set to 1 by ILBOSTG0 (for the STRING verb) or by ILBOUSTO (for the UNSTRING</p> | <p>verb) if an overflow condition occurs. This execution-time use is tested by generated code.</p> <p>7 Used for OPT. It is set to 1 by phase 62 if optimization has been requested for the program or by phase 65 if the SYMDMP, or STATE and OPT, or FLOW and OPT options have been specified.</p> <p>8 Reserved</p> <p>9 Used for CALL, CANCEL, or a recursive CALL. Set to 1 by the generated code for the CALL or CANCEL verb. Tested by INIT2 to determine if a recursive call condition exists.</p> <p>10 Execution-time STATE bit, relocated from bit 6 by ILBODBG0 and tested by the debugging subroutines.</p> <p>11 Set to 1 by the compiler if TEST option is in effect.</p> <p>12 QUOTE IS APOST bit. Set to 1 if the apostrophe is to be used to delineate literals and to be used in the generation of figurative constants.</p> <p>13 DBGFLPT bit. Set to 1 by phase 65 if, when SYMDMP is requested, there is a floating-point item in the program. Tested by ILBODBG0. Indicates that, following call to ILBODBG0, INIT3 contains DC of 2-byte displacement from beginning of PGT of virtual for ILBOTEF3.</p> <p>14 LONGTGT bit. Always set to 1.</p> <p>15 Indicates maximum length for a variable-length field. Before the execution of a Q-Routine, this bit is set to 1 if the VLC and SBL for the field are to be set to their maximum possible values, rather than a value depending on the current value of a data item. The maximum value is the value of X in the clause "OCCURS X TIMES DEPENDING ON...".</p> <p>16 SRVBIT bit. Indicates ILBOCOM subroutine is link edited with the object module.</p> |
|---|--|

- 17 Set to 1 if ENDJOB is specified.
- 18 Indicates that OBJECT-COMPUTER IBM-370 was specified. Used by COBOL Library subroutines.
- 19 Indicates that Q-routines are to be executed in initialization mode.
- 20 Set to 1 if COUNT is specified.
- 21 Set to 1 if the TRACE verb is specified.
- 22 Indicates that an invalid SYNADAF was performed in the ILBOSYN0 COBOL Library subroutine. Tested by code generated for Error Declarative.
- 24-31 DECIMAL-POINT IS COMMA clause byte. If this clause was specified, the byte contains a comma in EBCDIC. If not, it contains a decimal point.
- TALLY**  
a fullword used for source program references to the special register TALLY.
- SORT SAVE**  
a fullword used during the execution of a SORT/MERGE RETURN statement to contain the GN for the next sequential instruction following the RETURN.
- ENTRY-SAVE**  
a fullword used to save the entry point of the program during INIT2 and INIT3 execution.
- SORT CORE SIZE**  
a fullword for the SORT-CORE-SIZE special register as used in the source program.
- RET CODE**  
a halfword for the RETURN-CODE special register, which is used in the source program to provide a completion code on a STOP RUN, EXIT PROGRAM, or GOBACK statement, or to store the return code from a called program. It is the user's responsibility to set this code.
- SORT RET**  
a halfword used to contain the return code from a SORT/MERGE operation.
- WORKING CELLS**  
variable-length cells used by COBOL library subroutines called by the program. The total length of the field is 304 bytes.
- SORT FILE SIZE**  
a fullword for the SORT-FILE-SIZE special register as used in the source program.
- SORT MODE SIZE**  
a fullword for the SORT-MODE-SIZE special register as used in the source program.
- PGT-VN TBL**  
a fullword pointer to that part of the VN field of the PGT containing VNs for independent segments.
- TGT-VN TBL**  
a fullword pointer to that part of the VN field of the TGT containing VNs for independent segments.
- LENGTH OF VN TBL**  
a halfword containing the length of that part of the VN field (the length is the same for both the TGT and PGT) containing VNs for the independent segments.
- LABEL RET**  
the LABEL-RETURN special register for nonstandard labels. If an error occurs in such a label, it is the user's responsibility to place a nonzero value into this 1-byte cell.
- DDBG R14SAVE**  
contents of register 14. Phase 51 generates a call to ILBODBG4, the Save register 14 routine of the Debug control subroutine (ILBODBG0), before any instruction which passes control outside the COBOL program. The address of this instruction is saved in this cell by ILBODBG4.
- COBOL INDICATOR**  
identifies the object program as compiled by an OS/VSE COBOL or American National Standard COBOL compiler.
- A(INIT1)**  
address of INIT1 used for GOBACK, STOP RUN, and EXIT PROGRAM instructions, and for segmentation coding.
- DEBUG TABLE PTR**  
if the FLOW, STATE, SYNDMP, or TEST option is specified, this field contains displacement from the beginning of INIT1 to the Debug Table.
- SUBCOM PTR**  
address of the Subroutine communications area.
- SORT-MESSAGE**  
eight bytes for the SORT-MESSAGE special register which is used in the source program to allow the user to

indicate to the Sort/Merge program where to place the messages it issues.

**SYSOUT DDNAME**  
character which will be concatenated with "SYSOU" to create DISPLAY and TRACE EXHIBIT SYSOUT DDname. If field is zero, "T" is assumed.

**COBOL ID**  
a 2-byte field containing a binary number number that indicates the type and version of the compiler. The number is 0008 (hexadecimal) for the OS/VS COBOL compiler, release 1.

**COMPILED POINTER**  
an address (INIT1+88) pointing to the WHEN-COMPILED information.

**COUNT TABLE ADDRESS**  
relative address of the COUNT table from the beginning of the TGT. The COUNT table, which is generated by phase 60 or phase 64 if COUNT is specified, is located between the Q-routines, if any, and the INIT2 routine. The count table is used only when the program terminates.

**DBG R11SAVE**  
contents of register 11. When ILBODBG5, the Dynamic dumping routine of the Debug control subroutine (ILBODBG0) receives control, it places the return address to the inline code of the calling program in register 11. Therefore, the contents of register 11 must be saved before the call to ILBODBG5.

**COUNT CHAIN ADDRESS**  
address of the COUNT CHAIN for this program. The address is initialized to zero if count is specified; the address is filled in at execution time.

**PRBL1 CELL PTR**  
a fullword cell containing the address of the first PROCEDURE BLOCK cell in the PGT.

**TA LENGTH**  
a halfword initialized by phase 6 or 64 to the length of the largest segment with a nonzero priority.

**PCS LIT PTR**  
a fullword cell containing the address of the PCS (Program Collating Sequence) alphabet.

**DEBUGGING**  
a fullword cell containing the address of the beginning of the debugging cells in the variable portion of this table.

**CD FOR INITIAL INPUT**  
a fullword cell containing the address of the CD area with INITIAL INPUT clause.

**OVERFLOW**  
if the TGT is longer than 4096 bytes, this field contains one fullword cell pointing to each 4096-byte area after the first. The cell is loaded into a register when a base is required for the overflow area.

**BL**  
base locators. Each BL cell is a fullword containing an address in the data area. There is one BL pointing to the beginning of the Working-Storage Section and one for each file in the File Section. More than one BL is assigned if an area is larger than 4096 bytes. The BL assignments are made by phase 22. The BLs for queued files do not contain initial values and are changed by the execution of certain input/output operations.

**DECBADR**  
DECB addresses. There is one fullword cell pointing to the address of the DECB for each basic file.

**FIB**  
File Information Block addresses. There is one fullword cell pointing to the address of the FIB (address of the FCB during execution) for each VSAM file.

**DEBUG TRANSFER**  
a one byte cell indicating the DEBUG type for a PN.

**DEBUG CARD**  
a three-byte cell containing the card number.

**DEBUG BLL**  
a two-byte cell containing the displacement to the BLL cell.

**DEBUG VLC**  
a two-byte cell containing the displacement to the VLC cell.

**DEBUG MAX**  
a two-byte cell containing the maximum size of a DEBUG-ITEM.

**DEBUG PTR**  
a fullword cell containing a pointer used by ILBOBUG to reference the debug subscript table.

**TEMP STORAGE**  
temporary storage for arithmetic operations. TS space is allocated in doubleword blocks by phase 50. See

"Register and Storage Allocation" in the chapter "Phase 50."

**TEMP STORAGE-2**  
temporary storage for nonarithmetic instructions. These cells are variable in length. An example of the use of TS-2 may be found under "Nonarithmetic Conversions" in the chapter "Phase 51."

**TEMP STORAGE-3**  
temporary storage used to align fields of data described by the SYNCHRONIZED option. The field begins on a doubleword boundary.

**TEMP STORAGE-4**  
temporary storage cells used for the SEARCH ALL table-handling verb. The field starts on a doubleword boundary.

**BLL**  
base locators for the Linkage Section. Each BLL cell is a fullword containing the address of an area passed as a result of an ENTRY statement, a label record, a totaled area, a SORT description entry, or a GIVING option in a USE...ERROR statement.

**VLC**  
variable-length cells. Each VLC is a halfword whose value is set by the execution of a Q-Routine. It contains the current length of a variable-length field. There is one VLC for each OCCURS...DEPENDING ON clause and all items to which it is subordinate.

**SBL**  
secondary base locators. Each SBL cell is a fullword set by the execution of a Q-Routine. It contains the current address of a field which is variably located because it follows a variable-length field.

**INDEX**  
fullword cells, each containing the current value of an INDEX-NAME. There is one IND cell for each INDEX-NAME defined in an INDEXED BY clause.

**SUBADR**  
subscript addresses. Each SUBADR cell is a fullword containing the address for a subscripted reference. See "Resolving Subscripted and Indexed References" in the chapter "Phase 50."

**ONCTL**  
control counters for ON statements. Each is a fullword initialized to zero. See "Other Nonarithmetic Verb Strings" in the chapter "Phase 51" which discusses the ON statement.

**PFMCTL**  
PERFORM control counters and DEBUG saved location. Each PFMCTL cell is a fullword used for a PERFORM n TIMES statement to count the number of times the procedure has been performed. The instructions which use these cells are generated by phase 51. For DEBUG, a PFMCTL cell is used to save the contents of register 14 when the DEBUG packet is entered. DEBUG packets are called by BALR 14,15. See "PERFORM Statement" and "DEBUG Card" in the chapter "Phase 4."

**PFMSAV**  
PERFORM saved locations. Each is a fullword used to contain an address. The statements using these cells are generated by phase 4. For PERFORM, the cell is used to store the address of the next sequential instruction after the performed procedure, when that procedure is being executed because of a PERFORM. This is to enable the procedure to be executed inline.

**VN**  
variable procedure-names. Each VN cell is a doubleword containing the current address of a branch point which may change during program execution because of an ALTER or PERFORM statement. See "ALTER Statement" and "PERFORM Statement" in the chapter "Phase 4."

**SAVE AREA-2**  
pointer to the save area provided for label- and error-processing declaratives.

**SAVE AREA-3**  
variable number of fullwords used for OPEN parameters.

**XSASW**  
1-byte EXHIBIT switches. These are used as first-time switches for the coding generated for the EXHIBIT CHANGED statement. They are also used in certain types of SORT statements and ON statements.

**XSA**  
EXHIBIT saved area cells. These are variable in length and are referred to in the coding generated for an EXHIBIT CHANGED statement. There is one XSA for each operand to be exhibited with a CHANGED option. These cells are also used for SORT and RELEASE verbs.

**PARAM**  
parameter area of fullwords, containing parameter lists for macro instruction expansions of certain

source statements. The size of the parameter area equals the largest number of words required for any one expansion.

**RPTSAV**

six words used to save branch addresses during the execution of Report Writer routines, if the Report Writer is used.

**CHECKPT CTR**

fullword cells used to count the number of file records processed for a file for which checkpoints are to be taken.

**DEBUG TABLE**

this table is used by the flow trace statement number, and symbolic debug COBOL library subroutines. It is also used by the IBM OS COBOL Interactive Debug Program Product. It is built by phase 65; the format depends on the options specified.

- If the FLOW option is specified:

Byte	Contents
0	Number of traces requested.
1-3	Unused

- If the STATE option is specified:

Byte	Contents
0-3	Start of Q-Routines, or if none, start of INIT2.
4-7	Size of Declaratives (not including Report Writer) Section.
8-11	Address of PROCTAB in object module relative to beginning of INIT1.
12-15	Address of SEGINDX in object module relative to beginning of INIT1.
16-19	Address of end of SEGINDX in object module relative to beginning of INIT1.

- If both the FLOW and STATE options are specified:

Byte	Contents
0	Number of traces requested.
1-19	The same as shown above for the STATE option.

- If the SYMDMP or TEST option is specified:

Bytes	Contents
0-3	Start of Q-Routines or, if none start of INIT2.
4-5	Hashed compilation indicator (see PROGSUM table in "Section 5. Data Areas.")

- If the SYMDMP and FLOW options are specified:

Bytes	Contents
0	Number of traces requested
1-5	The same as shown above for the SYMDMP option.

**PROGRAM GLOBAL TABLE (PGT)**

The Program Global Table contains data which procedure instructions reference. All the fields in the PGT are variable in length. The data contained in the PGT is generated by phase 6 or 62. PGT data is never modified by procedure instructions; rather, it remains constant throughout program execution. In the chapter "Phase 6," the sections "Optimizing Storage for the Program Global Table" and "Allocating Storage for the Program Global Table" describe how the PGT is generated.

The fields in the PGT are illustrated in Figure 78 and described in the text below.

DEBUG LINKAGE AREA
COUNT LINKAGE AREA
TEST LINKAGE AREA
OVERFLOW
VIRTUAL
VIRTUAL EBCDIC NAMES
PN
GN
DCBADR
VNI
LITERAL
DISPLAY LITERAL
PROCEDURE BLOCK

Figure 78. Fields of the Program Global Table

**DEBUG LINKAGE AREA**

12-byte area which contains the linkage for dynamic dumps. The area contains the following code:

```
ST 11,DBG R11SAVE in TGT
L 11,=V(ILBODBG5)
```

BR 11  
(2 slack bytes)

If the SYMDMP option is not specified, this 12-byte area does not exist.

**COUNT LINKAGE AREA**

8-byte area which contains the linkage to the COUNT routine. If the COUNT option is not specified, this 8-byte area does not exist.

**TEST LINKAGE AREA**

16-byte field which contains the linkage to the IBM OS COBOL Interactive Debug Program Product (Program No. 5734-CB4) when TEST was specified for compilation. At the start of execution the field contains the following:

```
LR    14,0
BCTR  14,0
BCTR  14,0
MVI   0(14),X'07'
LR    14,0
BR    14
      (2 slack bytes)
```

**OVERFLOW**

if the entire PGT exceeds 4096 bytes in length, there is one fullword OVERFLOW cell pointing to each 4096-byte section after the first. The cell is loaded into a register when a base is needed to refer to the section of the PGT.

**VIRTUAL**

each virtual is a fullword containing the address of an external procedure (the result of an ESD and RLD in the object module) unless the DYNAM or RESIDENT option is in effect. If either option is in effect, the virtuals corresponding to library subroutines are written as EBCDIC

' / 00 00 00'

If the DYNAM option is in effect, the virtuals corresponding to user subprograms contain the relative displacement of the subprogram name from the beginning of the PGT. It is required because of a CALL statement in the source program or a branch to a COBOL library object-time subroutine. Because of phase 6 or 62 optimization, a given virtual is stored only once no matter how many times it is used.

**VIRTUAL EBCDIC NAMES**

EBCDIC names of library subroutines and user subprograms. If the DYNAM or RESIDENT option is in effect, the EBCDIC names of all library subroutines that are to be dynamically

loaded are listed; in addition, if DYNAM is in effect, the EBCDIC names of all user subprograms that are to be dynamically called are listed. Each VIRTUAL EBCDIC NAME cell is a doubleword containing the name of the subroutine or subprogram, left justified and padded with blanks if necessary. If neither DYNAM or RESIDENT is in effect, this field does not exist.

**PN**

source program procedure-names. When the OPT option is in effect, only those PNs associated with ALTER and declaratives references receive PN cells. Each PN cell is a fullword containing the address of the first instruction in a block of coding. The addresses of the PNs are in the same order as their definition in the source program. The program branches by loading an address from the PGT and then branching to it.

**GN**

compiler-generated procedure-names. When the OPT option is in effect, only those GNs associated with AT END and INVALID KEY references receive GN cells. Each GN is a fullword containing the address of the first instruction in a block of coding. GNs are used in the same way as PNs. They were generated to provide addresses for branches implied but not stated in the source program. They are stored in the PGT in the order in which they were generated.

**DCBADR**

DCB addresses. Each DCBADR cell is a fullword containing the address of a data control block in the data area of the program. There is one DCBADR cell for each DCB generated by phase 21.

**VNI**

variable procedure-name initialization cells. There is one doubleword VN cell for each variable procedure-name in the program. It contains the initial value of the VN, and is used to initialize the VN values in the TGT. VNs are generated by phase 4 to contain branch addresses which vary because of PERFORM or ALTER statements. See "PERFORM Statement" and "ALTER Statement" in the chapter "Phase 4" for a description of the generated logic.

**LITERAL**

literals referred to by procedure instructions. The literals are variable in length. There is no duplication in storage, since

duplicate literals were removed by phase 6 or 62 optimization.

**DISPLAY LITERAL**

literals used by calling sequences rather than instructions. They are variable in length. Duplication was eliminated by phase 6 or 62.

**PROCEDURE BLOCK**

each cell is a fullword containing the address of a Procedure block. The compiler assigns these cells only when the OPT option is in effect.

REPORT WRITER ROUTINES (RPT)

This area of the program contains the special routines which handle the Report Writer feature. It is discussed in "Appendix C. Report Writer Subprogram."

PROCEDURE

In this area is located the object code for the instructions described in the source program Procedure Division. The area does not contain any COBOL library subroutines; these routines, if required for execution, are added to the load module by the linkage editor or, if the RESIDENT or DYNAM option is in effect, these routines are dynamically loaded at object time. If the program is segmented, this area contains the Procedure Division sections for the root segment. See "Segmented Object Module" in this chapter.

The object code for this area is produced by phase 6 or 64, which also generates code for the Report Writer and Q-Routine areas. Note that phase 6 or 64 does not make any distinction among these three areas. They occur in the object module in this order because P0-text was written in that order: by phase 12 for Report Writer, by phase 1B for the Procedure Division, and by phase 22 for Q-Routines. From phase 3 on, the compiler did not recognize the end of one area and the beginning of the next.

Q-ROUTINES

Q-Routines are special routines generated for data items described by the OCCURS...DEPENDING ON option. The function of these routines is to update the length of the variable-length data item when that

length changes, and to update the location of the field which follows it. The actual output of a Q-Routine is a new value in the appropriate VLC cell of the TGT and the corresponding SBL cell (see the description of the TGT in this chapter for the meaning of these cells).

Only the Q-Routine updates the pointers; it does not change the contents of the data area involved. For this reason, if the OCCURS...DEPENDING ON area is followed by another field within the same 01-level item, and the OCCURS...DEPENDING ON area becomes longer, the information immediately following the area before it changed is now no longer accessible: the pointer to it, in the SBL, has been moved. This can be avoided in the source program by moving data out of the SBL field before any change in the value of the DEPENDING ON object, and moving it back after the change.

This problem does not arise between one 01-level item and the next, because each 01 field of data is allotted enough space for the maximum number of occurrences.

COUNT TABLE

The COUNT table is used by the COBOL library subroutine ILBOTC30. It contains entries for each procedure-name and source verb. This table is present only if COUNT was specified. The format of each entry is:

<u>Byte</u>	<u>Contents</u>
0	Identification code
	<u>Code</u> <u>Meaning</u>
	00 end of table
	01 procedure-id
	02 verb-id
1	Length of entry. If byte 0 contains 00, this byte contains zero.
*2-4	Card number
*5-6	Block number
	If byte 0 contains 01, these bytes contain zeros.
	If byte 0 contains 02, the block number of the count block for executable verbs; the count block is 00 for non-executable verbs.
*7	If byte 0 contains 02, the verb number (PL-code)
	or
*7 to n+1	If byte 0 contains 01, the EBCDIC name of the procedure



INITIALIZATION 2 ROUTINE (INIT2)

The initialization 2 (INIT2) routine is entered from INIT1 each time the program is executed. It is generated by phase 6 or 64. The routine performs the following functions:

1. Stores the address of the save area (located at the beginning of the TGT) for this program, which is contained in register 13, into the save area of the calling program or the job scheduler.
2. Stores the location of the save area of the calling program or job scheduler into this program's save area, which is located at the beginning of the TGT.
3. Stores the address of the first executable instruction in the ENTRY SAVE cell of the TGT in case register 14 is needed. INIT2 checks the SWITCH field of the TGT to determine if the program has been entered before. If so, it checks the SWITCH field to determine if this program has called another program and not yet received control back via a GOBACK or EXIT PROGRAM statement. This condition is known as a recursive call and causes the job step to be terminated. If INIT2 finds a recursive call, it passes control to ILBOSRV1 to issue an error message and terminate processing.
4. If neither the RESIDENT nor the DYNAM option is in effect, branches and links to the COBOL library subroutine ILBOSRV0 (see the publication IBM OS/VS COBOL Subroutine Library Program Logic to determine whether this is a main program or a subprogram in the run unit. Sets a bit in the SWITCH field of the TGT according to the information returned. If the DYNAM or RESIDENT option is in effect, INIT2 loads the COBOL library Subroutine communications area (SUBCOM) and stores the address in the TGT. It tests the SCHAIN flag in SUBCOM to determine if it is the main program. If it is a main program, INIT2 loads the COBOL library ENTER subroutine, ILBONTR0, and stores the address in SUBCOM. Otherwise, INIT2 issues a

-----  
 \*This field is not present if byte 0 contains 00.

DELETE macro instruction for SUBCOM to reduce the system responsibility count. If ILBONTR0 has not yet been loaded, INIT2 loads it and saves the address in SUBCOM. Then INIT2 obtains the address of ILBONTR0 from SUBCOM, sets the SCLDLST flag in SUBCOM, and passes control to ILBONTR0 to load all necessary COBOL library subroutines.

INITIALIZATION 3 ROUTINE (INIT3)

The initialization 3 (INIT3) routine is executed whenever the program is entered. It is generated by phase 6 or 64. The functions of the routine are determined by testing a bit in the SWITCH field of the TGT to determine whether this is the first time the module has been entered. If it is, the INIT3 routine:

1. Initializes the VN cells in the TGT from the values contained in the VN cells of the PGT.
2. Relocates each address constant in the TGT and PGT to its absolute location. (Until this routine is executed, the addresses are relative to the beginning of the program.) This relocation is accomplished by adding the absolute location of the beginning of the program (the address of INIT1, which is in register 11) to the relative addresses.
3. Loads and BALRs to the addresses of Q-Routines to initialize the VLC and SBL cells in the TGT for OCCURS...DEPENDING ON fields that depend on an item in Working-Storage.
4. Loads permanent base registers from BL cells in the TGT, or if the OPT option is in effect, loads permanent base registers from BL, BLL, and OVERFLOW cells in the TGT and PGT. (The meaning of permanent base registers is explained in the chapter "Phase 6" under "Execution-Time Base Register Assignment.")
5. Branches to the first executable instruction in the object program (at location START, whose address is in register 14), or to the entry point in the coding generated for an ENTRY statement, which is the instruction following the BALR in the ENTRY coding. If the FLOW, STATE, SYMDMP, or COUNT option is in effect, the COBOL library debugging subroutine, ILBDBGO, is loaded and branched to before the branch to the instruction. If TEST was specified at compile time,

INIT3 determines if the TEST command has been issued for execution time. If it has, INIT3 branches to the IBM OS COBOL Interactive Debug Program Product (Program No. 5734-CB4). If, additionally, there is a floating-point item in the program, the branch to ILBODBG0 is followed by a DC of the 2-byte displacement from the beginning of the PGT where the virtual for the ILBOTEF3 subroutine is located.

If the testing of the SWITCH field reveals that the module was entered previously, INIT3 resets the program to its previous state by loading registers from the save area in INIT1. Then it branches as described in item 5, above.

PROCTAB Table (PROCTAB)

The PROCTAB table is used by the object-time COBOL library subroutine for the statement number option. It contains entries for all the card numbers and verb numbers in the COBOL program. The format of each entry is:

Byte	Contents
0-2	Card number and verb number. The verb number is contained in the low-order 4 bits of byte 2.
3-4	Displacement of the verb within the fragment.

SEGINDX Table (SEGINDX)

The SEGINDX table is used by the object-time COBOL library subroutine for

the statement number option. It contains an entry for each fragment of the program. The format of each entry is:

Byte	Contents
0	Zero.
1-3	Fragment address.
4-6	Relative address of first PROCTAB entry for this fragment.
7-9	Relative address of last PROCTAB entry for this fragment.

SEGMENTED OBJECT MODULE (TRANSIENT AREA)

In a segmented program, the segment initialization subroutine ILBOGDO is always among the routines present. This routine is described in the publication IBM OS/VS COBOL Subroutine Library Program Logic.

If the user did not specify any Procedure Division sections with a priority lower than SEGMENT-LIMIT (or 49 by default), the procedure area of the root segment contains code generated by the compiler.

There are two types of nonroot segments: fixed segments and independent segments. Fixed segments are always made available in the state in which correspond to the LANGLVL(n) option while independent segments have any GO TO statements reset to their initial values, if they have been modified by ALTER statements. The subroutine ILBOSGM0 performs this resetting when necessary. It reinitializes those VN cells in the TGT which apply to this segment from the corresponding VN cells in the PGT.

APPENDIX C: REPORT WRITER SUBPROGRAM

This chapter describes the Report Writer Subprogram (RWS), its structure, elements, and response to verbs in the Procedure Division. The RWS is generated by phase 12 from statements in the Report Section of the Data Division. The operation of phase 12 and associated activities in other phases are described in the chapter "Phase 12".

If the OPT option is specified, the code generated for the Report Writer Subprogram is optimized for procedure-name addressability and register usage by phases 62, 63, and 64 in the same manner as is the code generated for the other parts of the Procedure Division. The operation of phases 62, 63, and 64 is described in the chapters "Phase 62," "Phase 63," and "Phase 64."

STRUCTURE OF THE REPORT WRITER SUBPROGRAM (RWS)

Each RWS is a complete subprogram which, when executed, produces a report according to the specifications coded in one RD statement and its associated group and elementary items. The RWS has a fixed logical structure; that is, it contains fixed, parametric, and group routines in a prescribed order and quantity. In certain cases, dummy routines are inserted to maintain the structure. The RWS produced contains all linkages and exits needed. Figure 79 shows the logic of a Report Writer Subprogram. The coding in the boxes is intended to be indicative rather than comprehensive.

ELEMENTS OF A REPORT WRITER SUBPROGRAM (RWS)

An RWS includes data items and three types of routines: fixed, parametric, and group. The data items are assigned special data-names; these can be either COBOL words, which may be used in the source program, or nonstandard words, which may not. These routines and data-names are discussed later, together with the special internal Report Writer verbs generated by the compiler.

## FIXED ROUTINES

Fixed routines never vary in logical content. Phase 12 generates one and only one copy of each of them after all of the statements under an RD have been scanned. The three fixed routines are 1ST-ROUT, LST-ROUT, and WRT-ROUT.

1ST-ROUT Routine

The 1ST-ROUT routine causes the first headings to be printed by calling on the RPT-ROUT and the CHF-ROUT routines. Routine 1ST-ROUT is executed when either GENERATE report-name or the first occurrence of GENERATE detail-name is encountered.

LST-ROUT Routine

The LST-ROUT routine terminates the report. It causes the highest level control break (by setting CTL.LVL to 1) and then causes the final footings to be printed by calling routines CFF-ROUT and RPF-ROUT. Routine LST-ROUT transfers control to the LAST-ROLL routine, which provides return linkage to the main program. (LAST-ROLL is the name of a STORE instruction located just before the ROL-ROUT routine. It is not itself an RWS routine.) Control then falls through to the ROL-ROUT routine.

WRT-ROUT Routine

The WRT-ROUT routine writes a record from the output work area, RPT.RCD. It then moves blanks to CTL.CHR and to RPT.LIN. This routine contains two sets of coding if the program specifies two output files. The programmer may suppress printing of a line by coding "MOVE 1 TO PRINT-SWITCH".

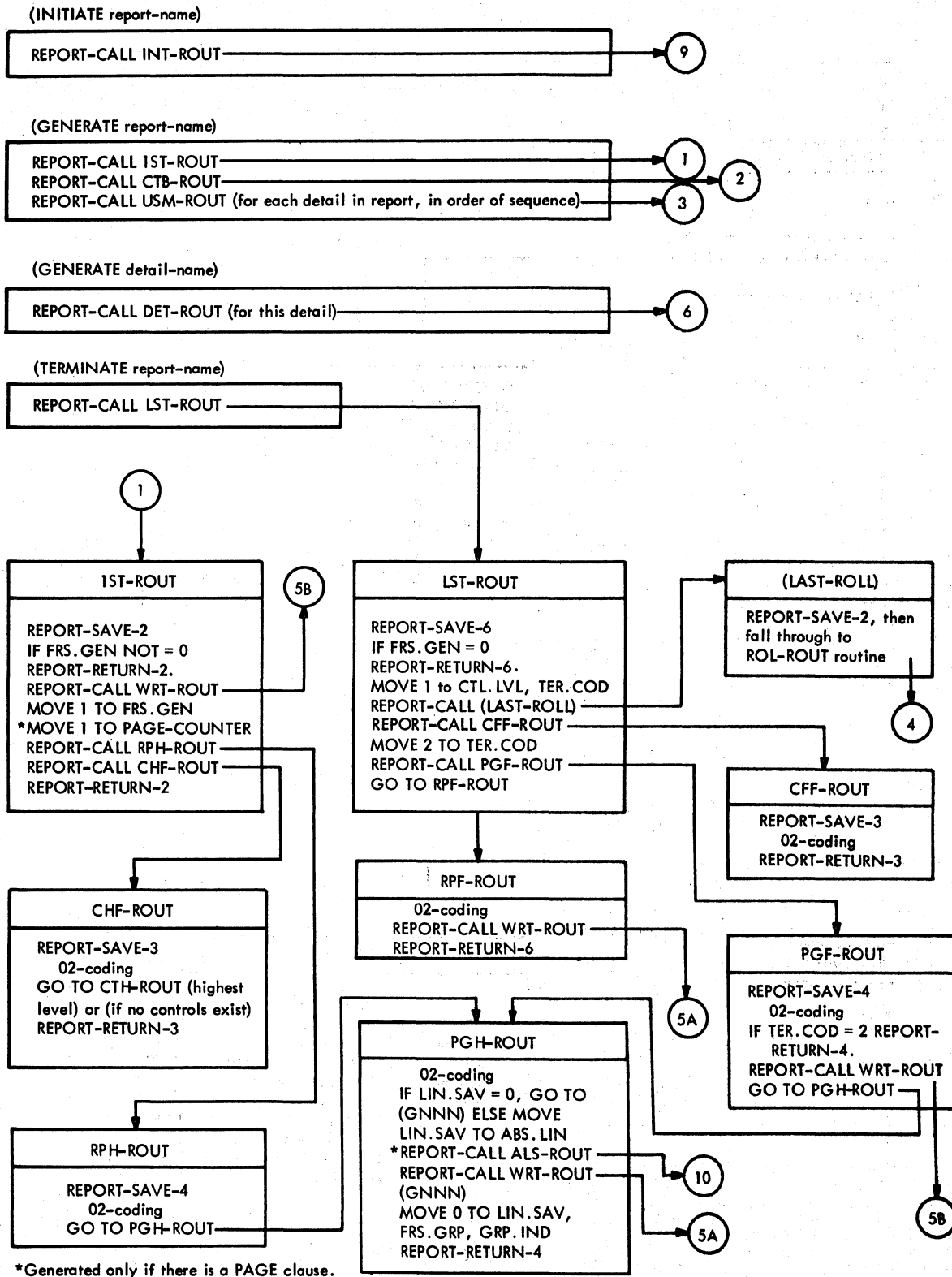
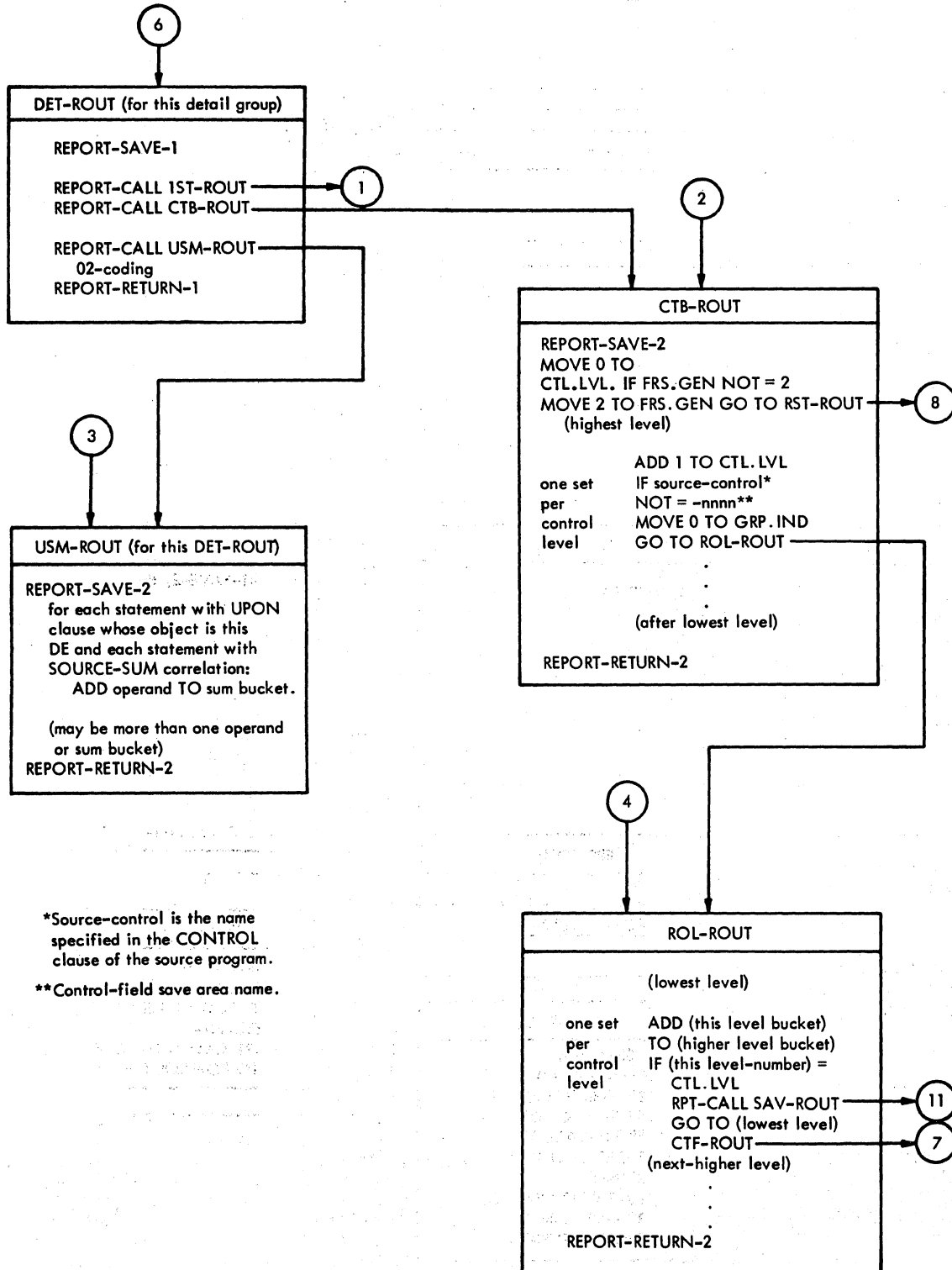


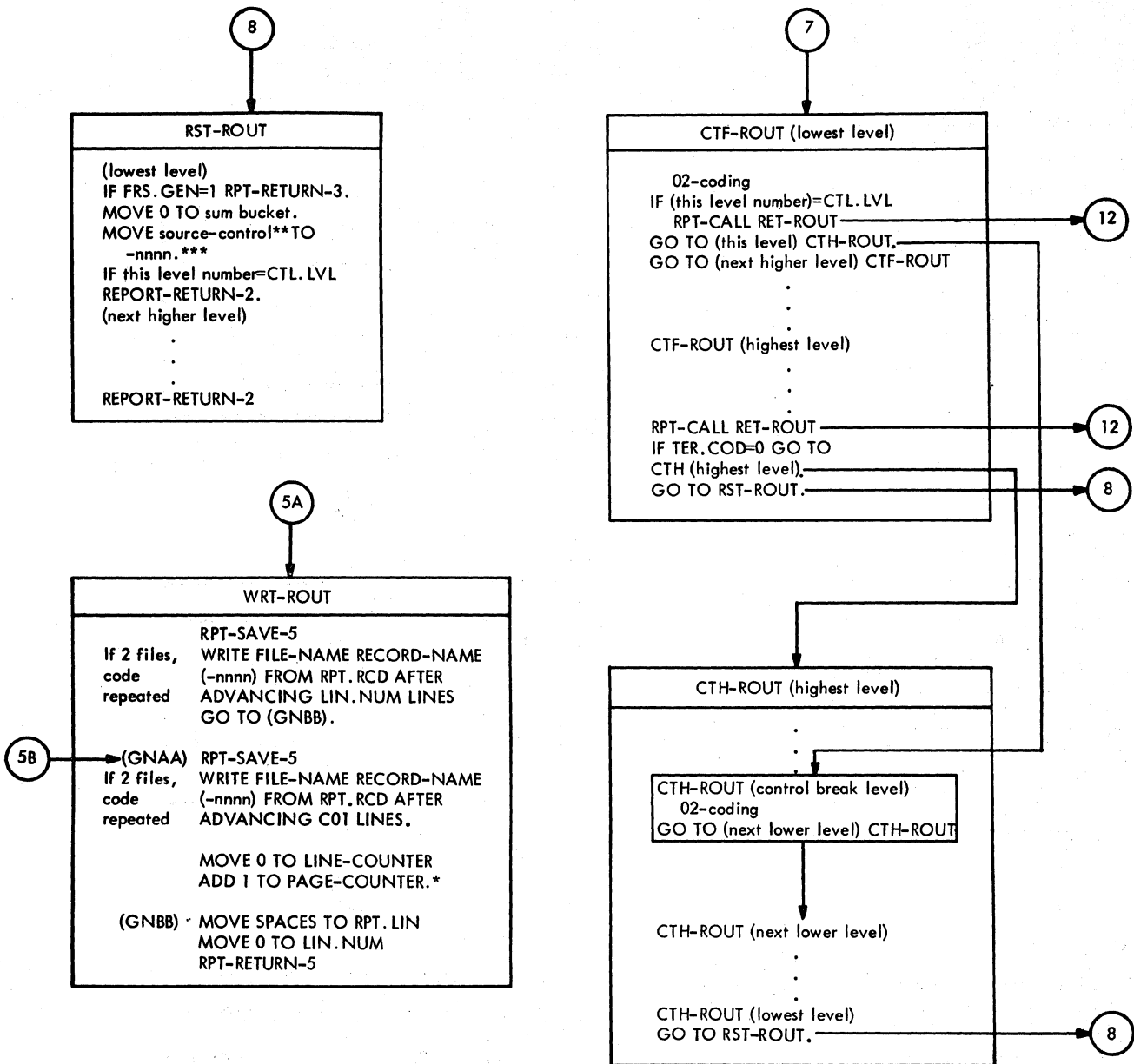
Figure 79 (Part 1 of 4). Logic of the Generated Report Writer Subprogram



\*Source-control is the name specified in the CONTROL clause of the source program.

\*\*Control-field save area name.

Figure 79. (Part 2 of 4). Logic of the Generated Report Writer Subprogram



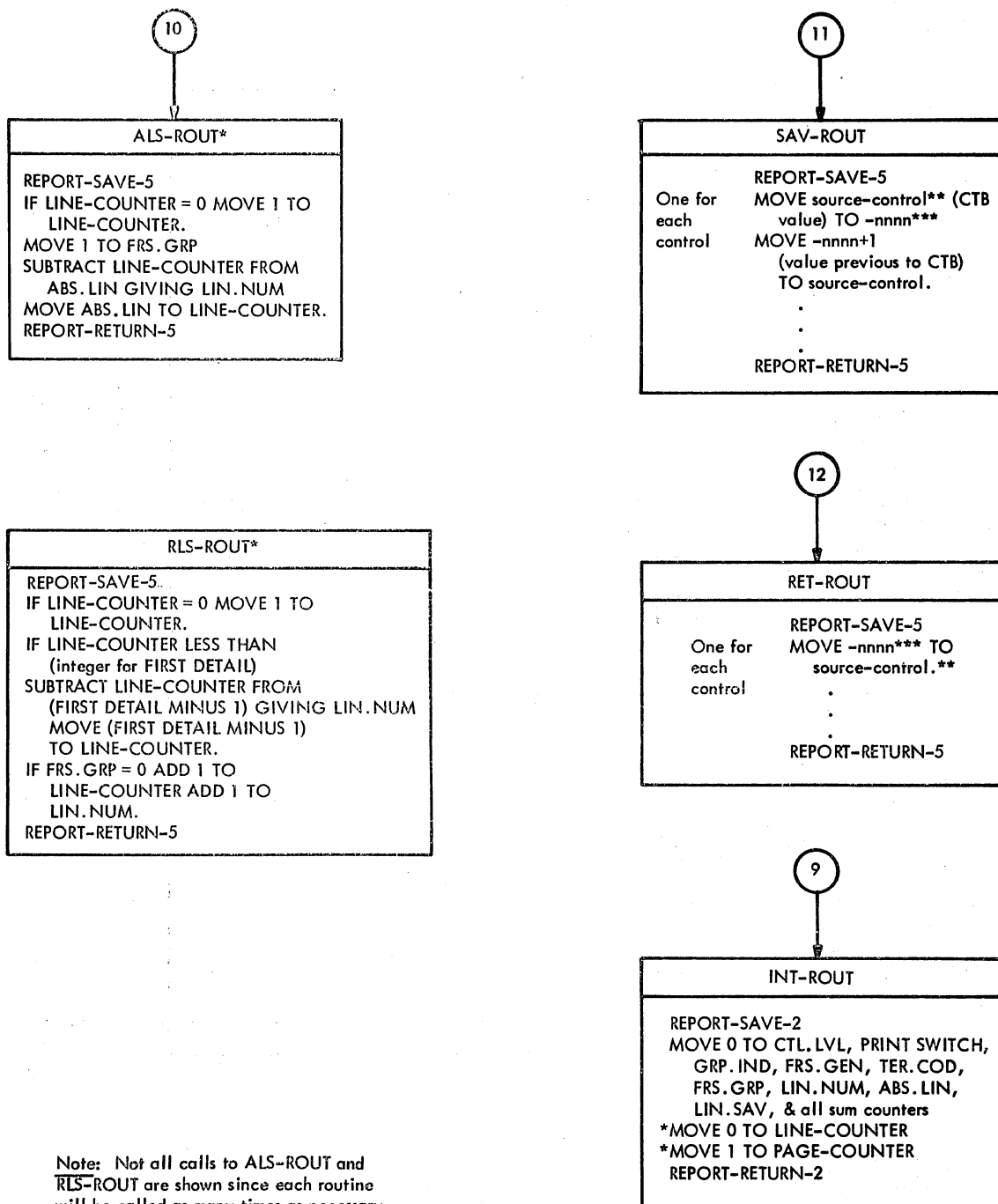
Note: All calls of the WRT-ROUT routine are not shown. Other routines call it as often as necessary to print all output lines to be produced.

\*Generated only if there is a PAGE clause.

\*\*Source-control is the name specified in the CONTROL clause of the source program.

\*\*\*Control-field save area name.

Figure 79 (Part 3 of 4). Logic of the Generated Report Writer Subprogram



Note: Not all calls to ALS-ROUT and RLS-ROUT are shown since each routine will be called as many times as necessary to determine line spacing.

\*Generated only if there is a PAGE clause.  
 \*\*Source-control is the name specified in the CONTROL clause of the source program.  
 \*\*\*Control-field save area name.

Figure 79 (Part 4 of 4). Logic of the Generated Report Writer Subprogram

## PARAMETRIC ROUTINES

Parametric routines generally are fixed in structure but vary according to the data obtained from the source (level-01 through level-49) statements. They may also include statements or blocks of statements that are repeated as needed. Except as noted under USM-ROUT, RLS-ROUT, and ALS-ROUT, the RWS contains one and only one copy of each parametric routine. The nine parametric routines are discussed in the following paragraphs.

### USM-ROUT Routine

The USM-ROUT routine adds the operands of all SUM clauses that either have UPON (this detail-name) or that appear as SOURCE items in a TYPE IS DETAIL group to as many sum buckets as required. A sum bucket is a work area that may be given a data-name by the programmer or else is assigned an S-point name by the compiler. (S-point names are described under "Nonstandard Data-names" in this appendix.) Phase 12 generates one USM-ROUT routine for each DET-ROUT routine. If there is no DET-ROUT routine in an RWS, no USM-ROUT routine is generated.

### CTB-ROUT Routine

The CTB-ROUT routine acts as a control break supervisor. It tests for a change in value of a control field, always beginning with the highest level and continuing until either the lowest level is tested or a control break occurs. A break causes control to be passed to the ROL-ROUT routine, leaving the current control level number in location CTL.LVL. The CTB-ROUT routine contains one block of coding for each control level.

### ROL-ROUT Routine

The ROL-ROUT routine adds SUM-clause operands originally defined in another control group. It starts with the lowest level and continues until the level number of the current block is equal to the value found in CTL.LVL. At this point, control is transferred to the lowest level CTF-ROUT routine. Routine ROL-ROUT contains one block of coding for each control level.

### RST-ROUT Routine

The RST-ROUT routine moves the current contents of sum buckets to control-field save areas and sets the sum buckets to zero for all control levels just processed by the ROL-ROUT routine. The RST-ROUT routine contains one block of coding for each control level. When the level number of the block being executed is equal to the value in CTL.LVL, control is returned to the routine that called the CTB-ROUT routine.

### SAV-ROUT Routine

The SAV-ROUT routine moves the current control name contents to a save area and the previous control name values to the current control names.

### RET-ROUT Routine

The RET-ROUT routine resets the control names to their current values.

Note: Routines SAV-ROUT and RET-ROUT are used only when processing TYPE IS CONTROL FOOTING or TYPE IS CONTROL FOOTING FINAL report groups. Therefore, any source control name areas contain the previous value (that is, the value prior to the control break).

### INT-ROUT Routine

The INT-ROUT routine sets initial values of all switches, counters, and SUM-names (data-names or S-point names). Routine INT-ROUT is called when an initiate statement is encountered.

### ALS-ROUT Routine

The ALS-ROUT routine determines the line spacing for absolute lines. The ALS-ROUT routine is generated only if the RD entry contains a PAGE LIMIT clause.



RLS-ROUT Routine

The RLS-ROUT routine determines the line spacing for relative lines. Routine RLS-ROUT is generated only if the RD entry contains a PAGE LIMIT clause.

there is neither an actual nor a dummy CTH-ROUT routine generated.

GROUP ROUTINES

Phase 12 generates one group routine for each level-01 record description encountered. The group routine selected is determined by the TYPE clause of the level-01 statement. The coding within the routine varies according to the level-01 through level-49 statements associated with it. A level-01 elementary item contains all necessary information and, hence, results in a complete routine.

If any of the group routines, except as discussed under DET-ROUT, CTH-ROUT, and CTF-ROUT routines, is not generated because there is no corresponding level-01 statement, phase 12 supplies a dummy routine to maintain the fixed logical structure of the RWS. The nine group routines are discussed in the following paragraphs.

CTF-ROUT Routine

The CTF-ROUT routine produces the control footings. There is one CTF-ROUT routine for each control (except FINAL) in the source program. It results from a TYPE IS CONTROL FOOTING group. If there is no such group, a dummy CTF-ROUT routine is generated for each control below the highest (FINAL) level. If, however, there are no controls (again, except FINAL), there is neither an actual nor a dummy CTF-ROUT routine generated.

CHF-ROUT Routine

The CHF-ROUT routine produces the heading for the highest (FINAL) level control. There is one CHF-ROUT routine in an RWS. It results from a TYPE IS CONTROL HEADING FINAL group. If there is no such group defined, or if there is no CONTROL clause in the program, a dummy CHF-ROUT routine is generated.

RPH-ROUT Routine

The RPT-ROUT routine produces the report heading. There is one RPH-ROUT routine in an RWS; it results from a TYPE IS REPORT HEADING group.

CFF-ROUT Routine

The CFF-ROUT routine produces the footing for the highest (FINAL) level control. There is one CFF-ROUT routine in an RWS. It results from a TYPE IS CONTROL FOOTING FINAL group. If there is no such group defined or if there is no CONTROL clause in the program, a dummy CFF-ROUT routine is generated.

RPF-ROUT Routine

The RPF-ROUT routine produces the report footing. There is one RPF-ROUT routine in an RWS; it results from a TYPE IS REPORT FOOTING group.

PGH-ROUT Routine

The PGH-ROUT routine produces the page headings. There is one PGH-ROUT routine in an RWS; it results from a TYPE IS PAGE HEADING group.

CTH-ROUT Routine

The CTH-ROUT routine produces the control headings. There is one CTH-ROUT routine for each control (except FINAL) in the source program. It results from a TYPE IS CONTROL HEADING group. If there is no such group, a dummy CTH-ROUT routine is generated for each control below the highest (FINAL) level. If, however, there are no controls (again, except FINAL),

PGF-ROUT Routine

The PGF-ROUT routine produces the page footings. There is one PGF-ROUT routine in an RWS; it results from a TYPE IS PAGE FOOTING group.

### DET-ROUT Routine

The DET-ROUT routine produces a detail line (or group of lines) of the report. There is one DET-ROUT routine for each TYPE IS DETAIL group. If there is no such group in the source program, there is neither an actual nor a dummy DET-ROUT routine generated.

### DATA-NAMES

Report Writer data-names are generated to identify counters, switches, and control fields. There are two types of data-names used in an RWS, COBOL word data-names and nonstandard data-names.

### COBOL Word Data-names

The COBOL word data-names follow the rules for coding COBOL names and are accessible to the source programmer. They are PAGE-COUNTER, LINE-COUNTER, and PRINT-SWITCH.

**PAGE-COUNTER:** A counter generated only if there is a PAGE LIMIT clause in the RD entry. There can be only one PAGE-COUNTER in an RWS. If present, it is initialized to 1 by the INT-ROUT routine and used by the WRT-ROUT routine.

**LINE-COUNTER:** A counter generated only if there is a PAGE LIMIT clause in the RD entry. There can be only one LINE-COUNTER in an RWS. If present, it is initialized to zero by the INT-ROUT routine and reset to zero by the WRT-ROUT routine for each new page.

**PRINT-SWITCH:** A 1-byte switch generated by phase 12 for any program that contains a Report Section. (It may then be used by any RWS generated for that program.) It is set to 0 by the INT-ROUT routine, indicating that the current line is to be printed. The source programmer can use PRINT-SWITCH to suppress printing of a report group by coding "MOVE 1 TO PRINT-SWITCH".

### Nonstandard Data-names

The nonstandard data-names contain the special character "." or begin with a hyphen; they cannot, therefore, be used by the programmer. Data-names in the form,

"- .nnnn" (for example, E.0001) and control-filed save area names have no limit and are uniquely numbered; the other nine appear once per report. The nonstandard data-names are:

**CTL.LVL:** A counter used by the CTB-ROUT, ROL-ROUT, CTF-ROUT, and RST-ROUT routines to coordinate control break activities. It is initialized to 0 by the INT-ROUT routine and set to 1 by the LST-ROUT routine.

**FRS.GEN:** A 1-byte switch used by the 1ST-ROUT and CTB-ROUT routines to ensure that routine 1ST-ROUT is executed once only. After the 1ST-ROUT routine is finished, FRS.GEN has a value of 1; after routine CTB-ROUT is executed, the value is 2. FRS.GEN is also tested by routine LST-ROUT to determine whether a TERMINATE was coded without an earlier GENERATE.

**GRP.IND:** A work area consisting of 1-byte switches. There is one switch for each GROUP INDICATE clause in a TYPE IS DETAIL group. The switches are turned on by the CTB-ROUT routine and individually tested by DET-ROUT routines after control or page break activities so that items specified in a GROUP INDICATE clause will be moved to the output line work area. The switches may be treated as a group or individually, as follows:

- GRP.IND: Group name (level-01) for a set of GP.nnnn names. It is set to 0 after a page or control break by the PGH-ROUT or the CTB-ROUT routine.
- GP.nnnn: Elementary names (level-02) following the GRP.IND. They are tested and, if zero, set to 1 by the DET-ROUT routine for a specific TYPE IS DETAIL group. Each GP.nnnn represents one 1-byte switch.

**TER.COD:** A 1-byte switch tested by the PGF-ROUT routine to prevent printing of an extra page heading, and by the CTF-ROUT routine (highest level) to determine if control headings should be produced. It is initialized to 0 by the INT-ROUT routine and set to 1 by the LST-ROUT routine.

**RPT.RCD:** The work area for the record containing the output print line. It is 133 bytes long and consists of either two or three parts (CODE-Cell is optional) in the following order: CODE-Cell, a 1-byte cell used to hold the code specified in the CODE clause of the RD statement and defined in the SPECIAL-NAMES paragraph; CTL.CHR, a 1-byte cell used to hold the carriage control character; and RPT.LIN, which contains the actual output print line. Note that, if there is no CODE clause there will be no CODE-Cell and RPT.LIN will be

132 bytes. The equivalent COBOL coding for the RPT.RCD group would be:

```
01 RPT.RCD.
   02 FILLER PICTURE X VALUE code.
   02 CTL.CHR PICTURE X VALUE SPACE.
   02 RPT.LIN PICTURE X(131) VALUE SPACE.
```

**ABS.LIN:** A 2-byte counter used by the ALS-ROUT routine for absolute line spacing. It is initialized to 0 by the INT-ROUT routine and set to the appropriate value as report lines are produced. It is set, therefore, by all group routines generated as a result of source statements, but not by dummy group routines.

**LIN.SAV:** A 2-byte save area. It contains either zero or an absolute line to be skipped to after a page heading is produced. If a Control Footing, Control Heading, or Detail report group contains a NEXT GROUP IS integer clause and if, after the presentation of that report group, the value of integer is less than or equal to LINE-COUNTER, then the integer is saved in LIN.SAV and the report group will space up to and including FOOTING.

**LIN.NUM:** A work area used in the WRT-ROUT routine in conjunction with the WRITE AFTER ADVANCING...LINES clause. LIN.NUM can be set by any group routine or by either the ALS-ROUT or the RLS-ROUT routine. Routine WRT-ROUT fills in LIN.NUM with zeros before exiting.

**FRS.GRP:** A switch set to 0 after the PGH-ROUT routine is executed. It is tested and set to 1 by a CTH-ROUT, CTF-ROUT, or DET-ROUT routine. If one of these groups is to be printed and if its first line is relative (that is, LINE PLUS integer), and if FRS.GRP is 0, the first relative line will be printed on either FIRST DETAIL or (LIN.SAV + 1).

**Control-field Save Area Names:** Data-names in the form "-nnnn" are names of control-field save areas. (There are two save areas per control level.)

A "-nnnn" name is also generated for any FD that contains a REPORT clause. The size of the level-01 item is determined from the RECORD CONTAINS clause or is 133 characters by default.

**E-point Data-names:** Data-names in the form "E.nnnn" are generated from COLUMN clauses in elementary record descriptions. They use the special RW-redefines of "RPT.LIN + COLUMN - (integer-1)".

**N-point Data-names:** Data-names in the form "N.nnnn" are counters used to hold the number of lines in a report group that contains a relative NEXT GROUP clause, at

least one relative LINE clause, or both. Using the N-point counter, the initial coding for a report group determines whether there are enough lines left on a page to print the entire group.

**S-point Data-names:** Data-names in the form "S.nnnn" are used for accumulators (sum buckets) for Control Footing record descriptions that have a SUM clause but no data-name specified. They are generated so that coding of MOVE sum bucket TO E.nnnn can be produced. Attributes of the SUM clause are picked up in the normal manner except for the PICTURE which is picked up from the corresponding E.nnnn name generated for the sum bucket. If the statement has a data-name, S.nnnn is not generated. However, its PICTURE is picked up in the same manner as an S.nnnn name.

#### SPECIAL REPORT WRITER VERBS

Phase 12 generates five special verbs for use in the RWS: REPORT-CALL, REPORT-SAVE, REPORT-RETURN, REPORT-ORIGIN, and REPORT-REORIGIN. The first three of these are used for linkage between the main program and the RWS -- for example, as a result of a GENERATE statement -- and between routines of the RWS itself. Their equivalent assembler language coding is shown below. The remaining two verbs are used to process USE BEFORE REPORTING sentences. In the following descriptions, the P0-text and P1-text verb codes are shown in parentheses after each verb.

**REPORT-CALL (4F):** The equivalent coding is:

```
L    15,A(Called routine)
BALR 1,15
```

**REPORT-SAVE-0 through REPORT-SAVE-n (50-55):** The equivalent coding is:

```
ST    1,Save-cell-n
```

**REPORT-RETURN-0 through REPORT-RETURN-n (56-5B):** The equivalent coding is:

```
L    1,Save-cell-n
BCR  15,1
```

**REPORT-ORIGIN (5C):** The execution of this verb causes the address counter to be set to the address of the RW-NOP statement at the start of the specified routine. A link to the USE routine is inserted at this point.

**REPORT-REORIGIN (5D):** The execution of this verb causes the address counter to be reset to the address it contained before the REPORT-ORIGIN was encountered.

RESPONSE TO PROCEDURE DIVISION VERBS

Once the Report Writer Subprogram has been generated, it is called at particular entry points and executed as a result of INITIATE, GENERATE, and TERMINATE statements in the Procedure Division of the source program. These responses are as follows:

Response to INITIATE: As a result of INITIATE, a branch is made to the INT-ROUT routine of the particular report. Routine INT-ROUT is executed and control returns to the next instruction after the INITIATE.

Response to GENERATE: The response to a GENERATE statement depends on whether the statement is the first such GENERATE or a subsequent one. Figures 80 and 81 illustrate the two cases. The logic flow shown is that for GENERATE detail-name statements. The logic for GENERATE report-name statements is the same except that all DET-ROUT routines are skipped and all USM-ROUT routines, in the order of their DET-ROUT routines, are executed.

Response to TERMINATE: The response to a TERMINATE statement is illustrated in Figure 82.

FINDING THE ELEMENTS OF A REPORT WRITER SUBPROGRAM (RWS)

It may become necessary to locate, in the object module or in a storage dump, the data items and routines that make up the RWS. This can best be done using a listing that includes a glossary and a cross-reference dictionary. The following discussion assumes the use of the DMAP and SXREF or XREF options.

LOCATING DATA ITEMS IN A STORAGE DUMP

The glossary lists the cells, switches, and work areas mentioned under "Data-names" in this appendix. A portion of the four pertinent columns of a typical glossary look, for example, like this:

SOURCE NAME	BASE	DISPL	INTRNL NAME
.	.	.	.
.	.	.	.
CTL.LVL	BL=3	088	DNM=2-426
.	.	.	.
.	.	.	.

To find cell CTL.LVL, turn to the memory map and find the BL cells in the TGT. BL1 is located at the address listed there and, 8 bytes farther, BL3. To the contents of BL3 add the displacement (DISPL), 88. The result is the address of CTL.LVL.

Note that if there are registers available for each BL needed in the program, one register is assigned permanently to BL3 and listed in the REGISTER ASSIGNMENT column of the memory map. In that case, add the DISPL to the contents of that register.

LOCATING DATA ITEMS IN THE OBJECT MODULE

To find references to a data item in the object module, note its internal name in the glossary and refer to the cross-reference dictionary. A portion of the cross-reference dictionary would look like this (again using CTL.LVL as the example):

DATA NAMES	DEFN	REFERENCE
.	.	.
.	.	.
CTL.LVL	0052	00100 00118
.	.	.
.	.	.
.	.	.

To the left of the object module appear the numbers of the source statements that generate each section of code; to the right, in the remarks column, are the internal data-names. Among the instructions generated for source statements 00100 and 00118 will be found references to item "DNM=2-426", the internal name for CTL.LVL.

LOCATING ROUTINES IN A STORAGE DUMP

To locate RWS routines in storage, identify the desired routine in the object module (discussed below), add the relative address to the load address (shown in the Linkage Editor map), and proceed as in finding any other instruction or routine.

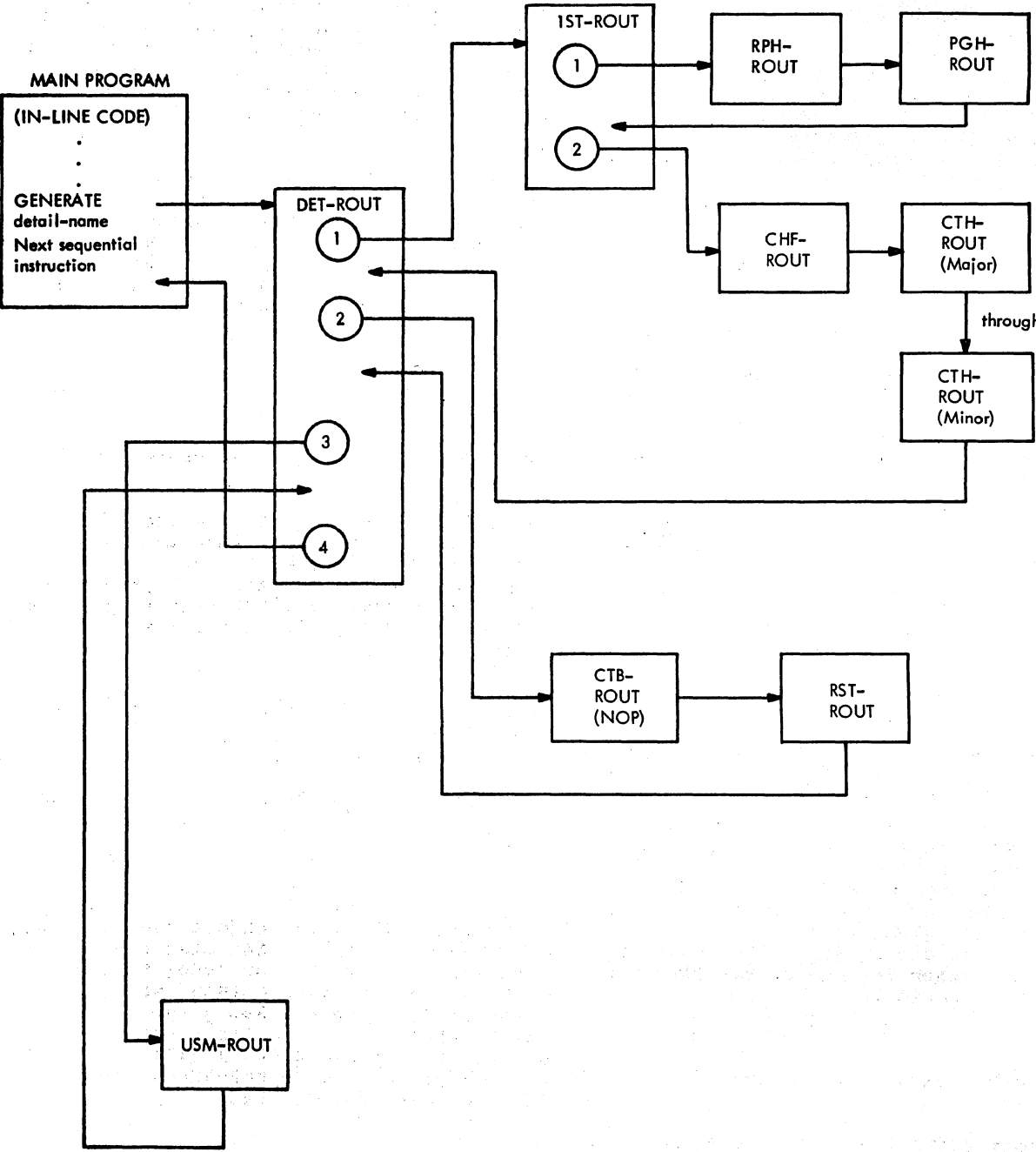


Figure 80. First GENERATE Statement Logic Flow

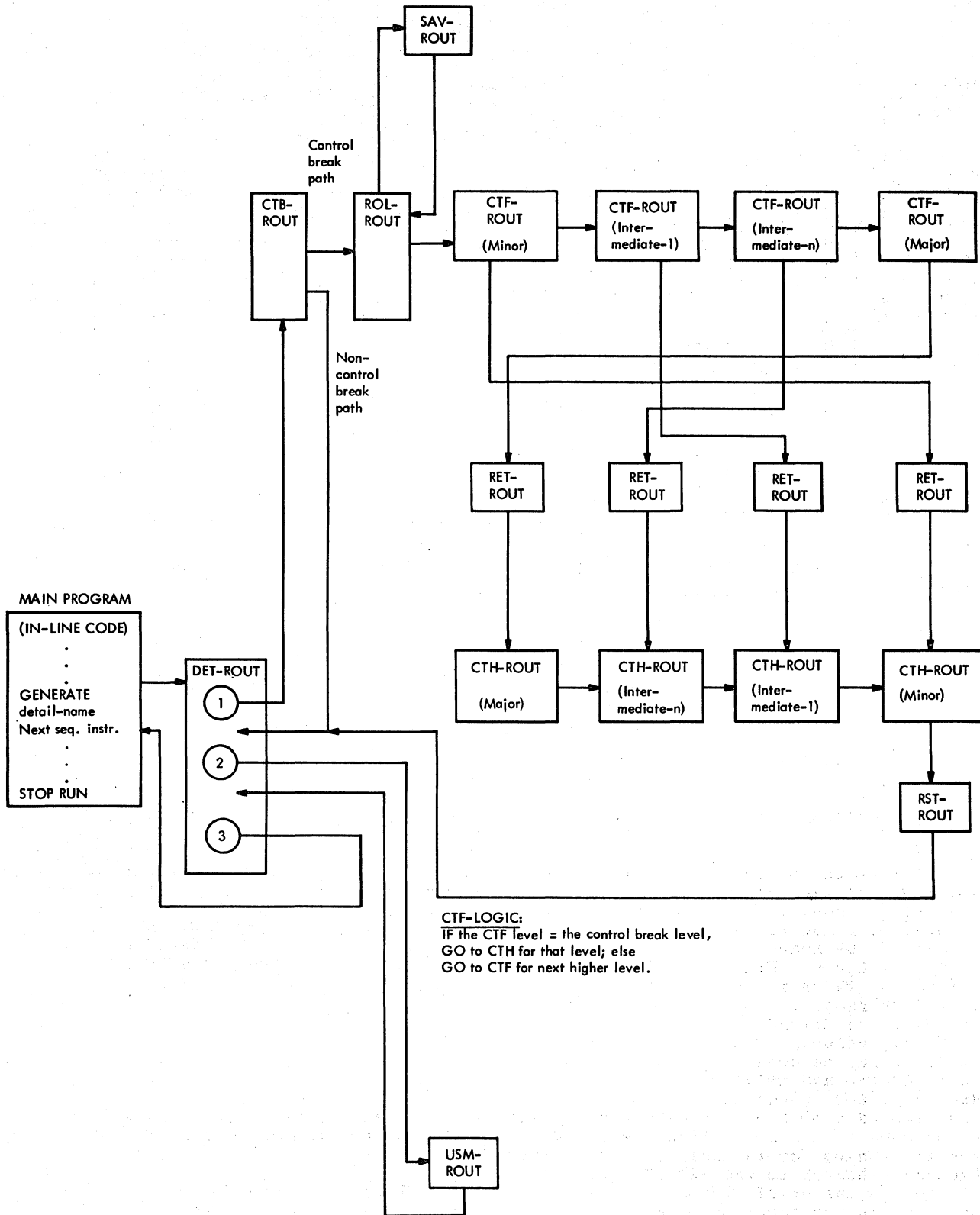


Figure 81. Logic Flow of All GENERATE Statements After the First

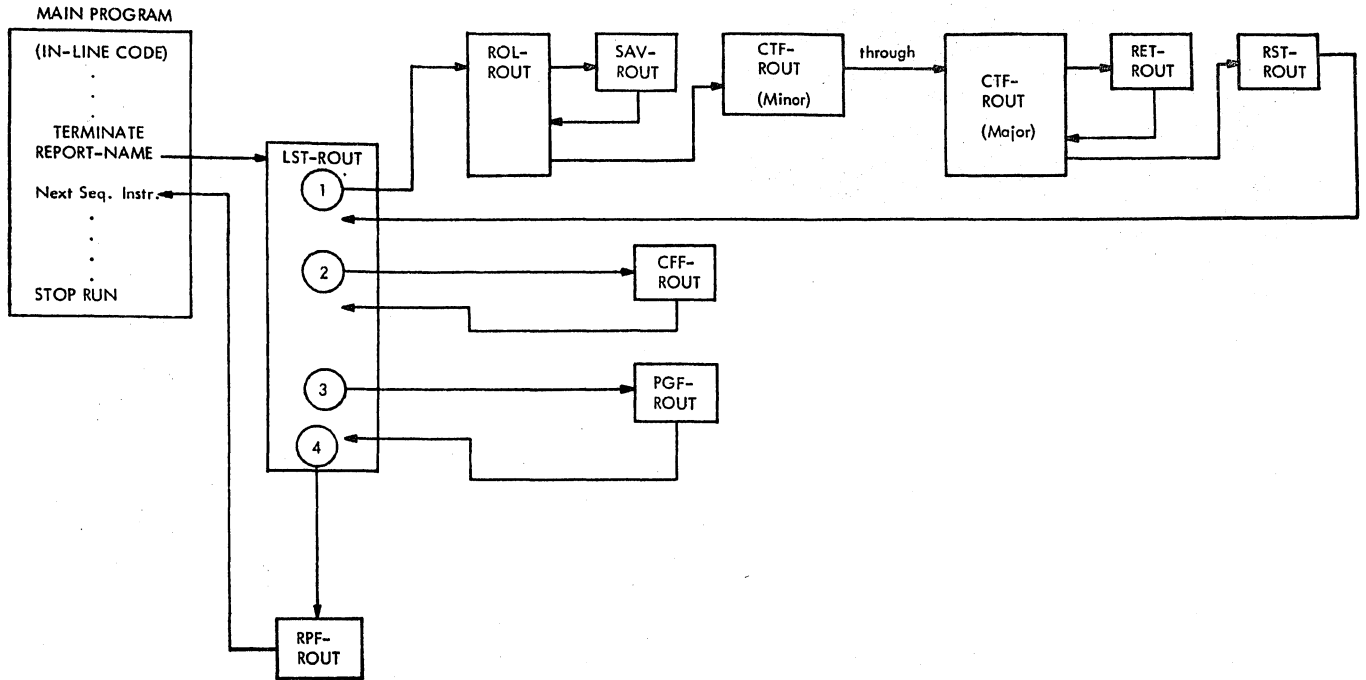


Figure 82. TERMINATE Statement Logic Flow

LOCATING ROUTINES IN THE OBJECT MODULE

RWS routines can be found by scanning the name field of the object module for their GN numbers. Most of their GN numbers can be found by using Figure 83. Phase 12 reserves 24 GN numbers while scanning each RD statement and assigns 17 of them to routines as shown in this table. (The other four routines, DET-ROUT, USM-ROUT, CTH-ROUT, and CTF-ROUT, are discussed separately below.) The GN numbers in Figure 83 may be considered absolute for the first RWS and relative for any succeeding RWSs generated. In the latter case, the GN number of the INT-ROUT routine can be used as a base. It may be found from the coding for the INITIATE statement, which is a branch to the INT-ROUT routine, with the GN number of that routine indicated in the remarks column.

Locating DET-ROUT and USM-ROUT Routines

There is one DET-ROUT routine generated for each detail group in the source program. Each DET-ROUT routine has one corresponding USM-ROUT routine. The DET-ROUT routines can be found by tracing from the level-01 statement containing the TYPE IS DETAIL clause. The generated instruction would probably be:

```
GN=032 EQU *
```

This is the first instruction of the DET-ROUT routine and 032 is the GN number.

Each DET-ROUT routine has one corresponding USM-ROUT routine. The USM-ROUT routine is assigned a GN number one less than its DET-ROUT routine, in this case 031.

GN	ROUTINE
01	RPH-ROUT
02	RPF-ROUT
03	PGH-ROUT
04	PGF-ROUT
05	1ST-ROUT
06	LST-ROUT
07	WRT-ROUT
010	CTB-ROUT
011	ROL-ROUT
012	RST-ROUT
016	CHF-ROUT
017	CFE-ROUT
020	INT-ROUT
021	ALS-ROUT
022	RLS-ROUT
023	SAV-ROUT
024	RET-ROUT

Figure 83. Report Writer Subprogram GN Numbers

Locating CTF-ROUT and CTH-ROUT Routines

One CTF-ROUT and one CTH-ROUT routine are assigned to each control after the highest (FINAL) level control (whose heading and footing are provided by the CHF-ROUT and CFF-ROUT routines). If they are described in the source program, they may be found in the same way as the DET-ROUT routines. If not, they can be found by tracing the logic, using Figure 79 as a guide.



APPENDIX D: INTERFACE WITH CONVERSATIONAL MONITOR SYSTEM (CMS)

The COBOL-CMS Interface routine is a module executable under the Conversational Monitor System (CMS) which allows the terminal user to compile a COBOL source program by invoking the IBM OS/VS COBOL Compiler.

INTRODUCTION

The COBOL-CMS Interface routine allows the terminal user a conversational means of specifying the options, input and output data sets, and libraries to be used by the COBOL compiler. It also dynamically allocates all data sets for the user. It stores the compiled code for execution either under OS/VS or under control of CMS (for information on the interface between the compiled code and CMS see IBM OS/VS COBOL Subroutine Program Logic).

FUNCTIONS

The COBOL-CMS Interface routine analyzes a COBOL command string from the terminal, constructs an option list, issues CMS FILEDEF commands according to the options specified, and calls IKFCBL00 to compile the source file. The relationships among the DMSCOB routine, the COBOL compiler, and CMS are described in Figure 84.

ENVIRONMENT

The COBOL-CMS Interface routine, DMSCOB, (known to CMS as the COBOL command module) operates under the Conversational Monitor System (CMS) in the virtual machine environment of the IBM Virtual Machine Facility/370 (VM/370).

The COBOL-CMS Interface routine is initiated when the COBOL command is issued through a terminal, such as the 2741 Communications Terminal.

Input to the COBOL compiler is from the file specified in the filename parameter. This is the COBOL source program. Only the device types: DISK, READER, and TAPE, are valid as input source file devices. The FILEDEF command for a DDname of a COBOL source program must be issued if the source file resides either on tape or in the virtual reader. The default medium is DISK.

Diagnostic, informational, and prompting messages are typed at the terminal.

The LISTING file which is produced by the compiler may be written on disk, the spooled printer, or to a dummy device. The object (TEXT) file produced by the compiler may be written as a disk file, a spooled punched deck, or to a dummy device.

PHYSICAL CHARACTERISTICS

The COBOL-CMS Interface routine resides on the CMS system disk with the IBM OS/VS COBOL Compiler and is given control by the initialization routine of CMS. It consists of two object modules, DMSCOB and DMSCBD. DMSCOB constructs the option list, handles linkages to and from the compiler, and returns control at the end of compilation to the CMS command level.

DMSCBD contains directory information about each phase of the compiler. After installation, DMSCBD is physically part of the DMSCOB module.

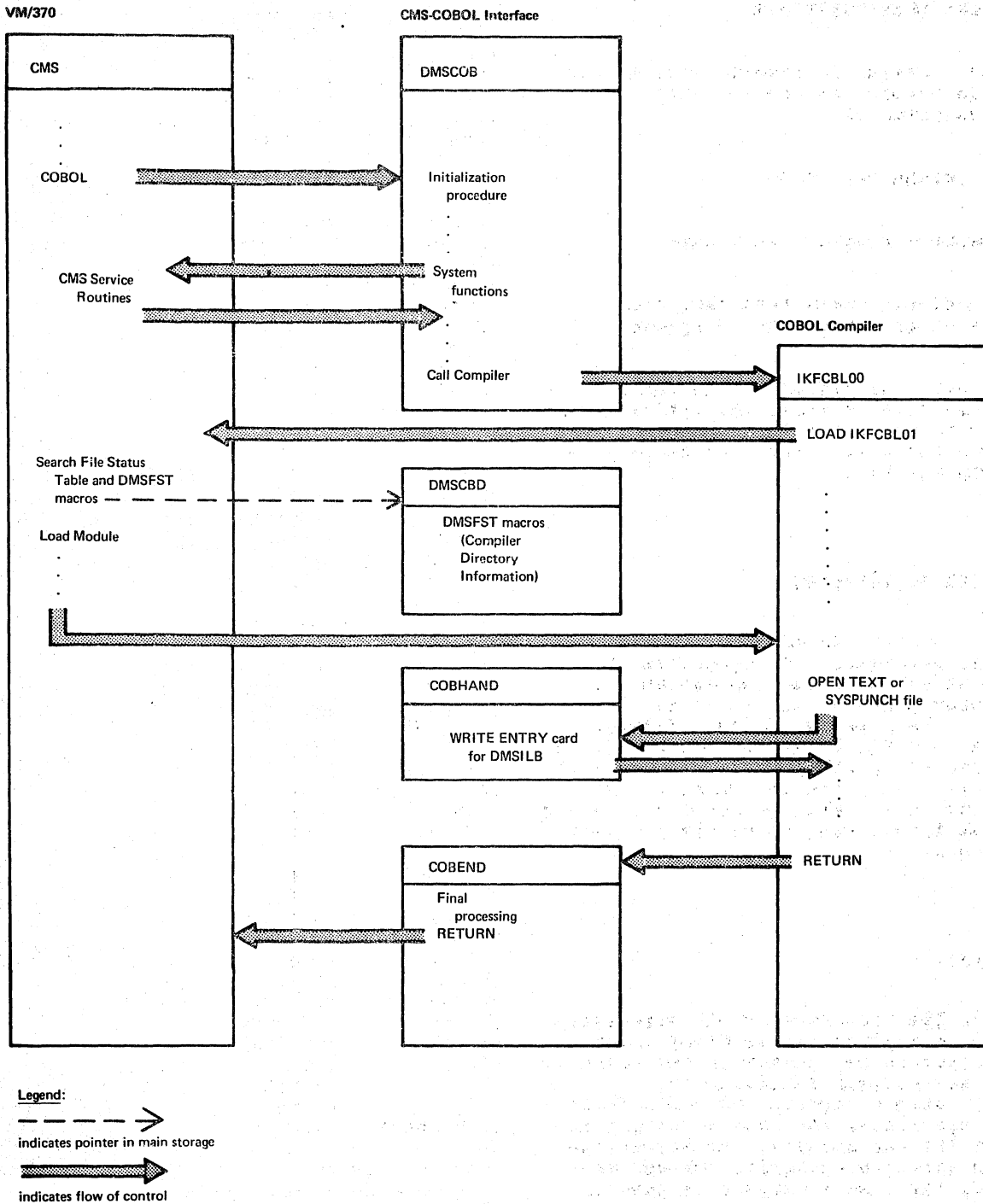


Figure 84. Relationships Among CMS-COBOL Interface Routine, the COBOL Compiler, and CMS

**OPERATIONAL CONSIDERATIONS**

The DMSCOB routine is invoked when a COBOL command is entered at the terminal. The command consists of:

- The command name COBOL
- The source program 'filename'
- Any optional parameters that the terminal user may wish to specify

The parameters specified in the COBOL command are used to build the option list and to issue the CMS FILEDEF commands needed for the output from the compiler and the DMSCOB routine.

Source Program Filename

The source program filename is a required positional parameter. It specifies the filename of the data set containing the COBOL source program that is to be compiled. The file must have a filetype of COBOL and have fixed-length records with a logical record length permissible for that device type. Standard search rules are followed to locate the specified file among those disk directories currently accessible to the user.

Option List

The option list consists of all parameters specified or implied in the COBOL command. These parameters may appear in any order in a set of parentheses following the filename. When an option and its default are both specified, the last to appear is generally the one assumed. If any of the following mutually exclusive options are specified, the last to appear is assumed:

- CLIST - PMAP
- XREF - SXREF
- PRINT - DISK

A maximum of 100 characters, including delimiter blanks, between options are allowed within the set of parentheses. The options are listed in Figure 85.

Option	Default	Alternate Names
NOSOURCE	SOURCE	NOSOU,SOU
CLIST	NOCLIST	CLI,NOCLI
DMAP	NODMAP	DMA,NODMA
PMAP	NOPMAP	PMA,NOPMA
NOLOAD	LOAD	NOLOA,LOA
DECK	NODECK	DEC,NODEC
NOSEQ	SEQ	
FLAGE	FLAGW	LAG,LAGW
SUPMAP	NOSUPMAP	SUP,NOSUP
TRUNC	NOTRUNC	TRU,NOTRU
SPACE2	SPACE1	ACE2,ACE1
SPACE3	SPACE1	ACE3,ACE1
NUM	NONUM	
QUOTE	APOST	QUO,APO
STATE	NOSTATE	STA,NOSTA
FLOWnn	NOFLOW	FLO,NOFLO
XREF	NOXREF	XRE,NOXRE
SXREF	NOSXREF	SXR,NOSXR
ADV	NOADV	ADV,OADV
NOTERM	TERM	NOTER,TER
NOLIB	LIB	
BATCH	NOBATCH	BAT,NOBAT
NAME	NONAME	NAM,NONAM
NOZWB	ZWB	
SYMDMP	NOSYMDMP	SYM,NOSYM
OPT	NOOPT	
RES	NORES	
DYNAM	NODYNAM	DYN,NODYN
SYNTAX	NOSYNTAX	SYN,NOSYN
CSYNTAX	NOSYNTAX	CSYN,NOSYN
PRINT*	DISK*	PRI,DI
NOPRINT*	DISK*	DI,NOPRI
SIZE	81920	SIZ YYY
YYYYYY		
BUF YYYYY	2768	BUF YYY
SYSx	SYST	
NOVERB	VERB	NOVER,VER
OSDECK		OSD
LSTCOMP	NOLST	LSTC,OLST
LSTONLY	NOLST	LSTO,OLST
CDECK	NOCDECK	CDE,OCDE
FDECK	NOFDECK	FDE,OFDE
LCOL1	LCOL2	OL1,OL2
L120	L132	L12,L13
COUNT	NOCOUNT	COU,OCO
VBREF	NOVBREF	VBR,OVBR
VBSUM	NOVBSUM	VBS,OVBS
NODUMP	DUMP	ODUM,DUM
LVL	NOLVL	LVL,OLVL
ADV	NOADV	ADV,OADV
LANGLVL (1)	LANGLVL (2)	
See Note below.		

Figure 85. COBOL Compiler Options Under CMS

**Note:** The PRINT, NOPRINT, and DISK options apply only in the CMS environment. An explanation follows:

- PRINT specifies that a program listing is to be produced. The listing includes page headings, line numbers of the statements in error, message

identification numbers, severity levels, and message texts (as well as any other output requested by SOURCE, CLIST, DMAP, PMAP, XREF, or SXREF). The listing is printed at the spooled printer.

- DISK specifies that a program listing (as described for PRINT above) is to be produced, but that it is to be written to the appropriate read/write disk with a filetype of LISTING instead of to the spooled printer.
- NOPRINT specifies that the listing file described for PRINT above is not to be written either to the spooled printer or to the read/write disk described for DISK.
- OSDECK specifies that the object program is to be executed under OS/VS. If OSDECK is not specified, it is assumed that the object program is to be executed under CMS.

The DISK option is not recognized by the compiler. The FDFLIST routine of DMSCOB processes this option by directing the compiler output specified for SYSPRINT to a disk file called LISTING.

The other options that appear in Figure 85 are described in "Compiler Options" in the chapter "Introduction."

Issuing CMS FILEDEF Commands

The DMSCOB module calls routine DMSFLD to issue FILEDEF commands. This command is used to specify the input/output devices and data set characteristics that are required according to the user specified options. These data sets are listed in Figure 86. For a description of CMS see IBM Virtual Machine Facility/370 Conversational Monitor System (CMS) Program Logic, Order No. SY20-0881.

METHOD OF OPERATION

The DMSCOB routine is invoked by the CMS command processor in response to a COBOL command from a terminal user. DMSCOB performs the initialization that is needed to call the IBM OS/VS COBOL Compiler.

After compilation is complete or, if any D- or E-level messages have been issued, after error processing is complete, DMSCOB receives control to close all files and return control to the CMS command environment.

Filetype/Filename	Condition Required	Comment
SYSUT1-SYSUT4	Always created	Erased at end of compilation.
SYSUT5	SYMDMP option	Written to disk where source file resides if that is read/write; if not, to its parent disk if that is read/write; or else to primary disk if that is read/write.
LISTING (SYSPRINT)	SOURCE, DMAP, PMAP, CLIST, XREF, SXREF, DISK options	
TEXT	No E- or D-level error messages produced	Machine-language code created by compiler; written to same disk as SYSUT5 and/or LISTING.
SYSLIB	LIB option	COPY or BASIS statements in source program.
SYSTEM	TERM option	Progress and diagnostic messages and compiler statistics written to the terminal.
SYPUNCH	DECK option	Object module is written on spooled punch.
SYSUT6	LVL option	Compiler SYSPRINT output passed on SYSUT6 to FIPS processor; erased at end of compilation.

Figure 86. FILEDEF Commands Issued for Compilation Under CMS

## INITIALIZATION

The operations of DMSCOB at initialization are described in Figure 87. Most of the processing is described in the "Extended Descriptions" section of the diagram. See the sections "Option List" and "Issuing CMS FILEDEF Commands" under "Operational Considerations" for further information. Additional explanations are given below for:

- Special processing for TEXT and SYSPUNCH files.
- Compiler directory information.
- Error processing.

### Special Processing for TEXT and SYSPUNCH Files

After the FILEDEF commands (described above) have been issued for the TEXT and SYSPUNCH files, the DMSCOB routine sets a flag in the CMS Control Block for each file to indicate that the auxiliary processing routine COBHAND is to be given control when these files are opened during compilation. The CMS OPEN procedure passes control to COBHAND after the OPEN has been completed.

When the COBHAND routine is called, it writes a CMS Loader ENTRY control card on the TEXT or SYSPUNCH file. The control card is the first card in the compiled program. It starts in column 1 and specifies that ILBCMS is the external name of the module that is to be loaded with the compiled program. This is the entry point of the execution-time COBOL-CMS Interface routine, DMSILB.

At execution time, when the user issues the LOAD command, the Loader loads the DMSILB routine along with the compiled COBOL program. When the user issues the START command, the DMSILB routine is given control to do the initialization necessary to run the compiled program under CMS.

(For a description of the execution-time COBOL-CMS interface, see IBM OS/VS COBOL Subroutine Library Program Logic. For a description of the CMS commands see IBM Virtual Machine Facility/370 Command Language User's Guide, Order No. GC20-1804.)

**Note:** If a program is compiled under CMS and the OSDECK option is specified, the ENTRY control card is not written in the object file. The program cannot subsequently be executed under CMS.

### Compiler Directory Information

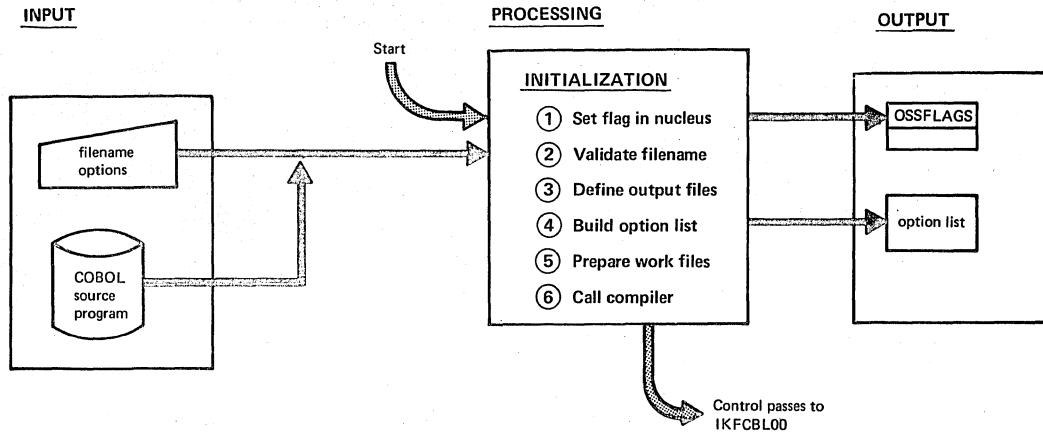
At installation time the DMSGND routine of CMS resolves the necessary directory information for each phase of the COBOL compiler. This information is stored in the form of a DMSFST macro instruction in the DMSCBD module. After initialization, DMSCBD is physically part of the DMSCOB module.

During compilation the CMS Loader searches the File Status Table and all DMSFST macros whenever a LOAD instruction is issued for a phase of the compiler. It uses this information to locate the phase.

For descriptions of the DMSGND routine, the DMSFST macro instruction, the CMS loader, and the File Status Table see IBM Virtual Machine Facility/370 Conversational Monitor System (CMS) Program Logic, Order No. SY20-0881.

### Error Processing

During initialization, DMSCOB issues error messages for user errors involving the filename parameter, the file containing the COBOL source program, or any of the options specified. The error messages issued by DMSCOB are listed in the section "Diagnostic Aids."



Extended Description	Label	Chart
① Set compile switch in OSSFLAGS located in NUCON (see <i>IBM Virtual Machine Facility/370 Conversational Monitor System (CMS) Program Logic</i> , Order No. SY20-0881 for the format of NUCON)	DMSCOB	VM01A1
② Check command line from terminal for filename. Call CMS control program to verify filename. Determine that file has fixed-length records.	CKFNAME STAT	VM01D1 VM01F1
③ Call CMS control program to define a disk for permanent output files. Use COBOL source program disk if it is read/write; if read only, use its parent disk if that disk is read/write; if read only, use primary disk if that disk is read/write.	FINDRW ANYRW USEIT	VM01J1 VM01G3 VM02G1
④ Build parameter list for specified compiler options in EBCDIC format. Set switches to control allocation of output devices. Flag invalid options.	OPTSCN	VM02B4
⑤ Call CMS control program to determine read/write disk with most space for utility files. Call CMS control program to erase any old TEXT, LISTING, and utility files with same filename as file being compiled. Call CMS control program to issue the CMS FILEDEF commands required by options specified.	FDEFS FDFEFLIST	VM03G1 VM03H1
⑥ Issue GETMAIN for storage needed by compiler. Set address of DMSCBD in File Status Table Extension. Call IKFCBL00 to compile source file.	NOOPT STDPLIST	VM03A1 VM03G2

Figure 87. Operations of DMSCOB Routine at Initialization

**RETURNING CONTROL TO THE CMS COMMAND ENVIRONMENT**

When DMSCOB receives control from IKFCBL00, it saves the return code issued by the compiler and issues a warning message for any nonzero return code. It then calls other modules to perform required functions. The routines and their functions are as follows:

<u>Routine</u>	<u>Function</u>
DMSFNSA	Close all files
DMSERS	Erase all utility files
DMSAUPD	Update the user file directory
DMSFLD	Clear all file definitions
DMSSTMN	Reinitialize storage

Then the routine resets switches, indicates that the virtual storage pages used by the compiler are no longer required and can be released, sets a return code, and returns to the CMS command environment. Linkages to these routines are described in "Linkages" in the section "Diagnostic Aids."

For descriptions of the routines see IBM Virtual Machine Facility/370 Conversational Monitor System (CMS) Program Logic, Order No. SY20-0881.

**PROGRAM ORGANIZATION**

The COBOL-CMS Interface routine consists of two modules, DMSCOB and DMSCBD. DMSCBD

contains all the executable code that is required for initialization and for transfer of control to the OS/VS COBOL Compiler. DMSCBD contains the directory information that is required by the Loader to locate the phases of the compiler. After system installation, DMSCBD is physically part of the DMSCOB module.

This section describes the organization of the Interface Routine. A standard flowchart and directory information are provided.

**DIRECTORIES**

This section contains four directories to be used in conjunction with microfiche listings of the interface routine.

Figure 88 associates load modules with the CSECTs they contain. Figure 89 associates external symbols with the load modules in which they appear. Microfiche names are usually the same as the load module names shown in these directories.

Figure 90 is a directory of all the labels that appear in the flowchart in this appendix.

<u>External Symbol</u>	<u>Type</u>	<u>Load/Object Module</u>
COBOL	LD	DMSCOB
DMSCBD	ER	DMSCOB (after installation)
DMSCOB	SD	DMSCOB
NUCON	ER	NUCON

Figure 89. External Symbol Directory

<u>Load Module</u>	<u>Entry</u>	<u>CSECT Names</u>	<u>Function</u>
DMSCBD	Non-executable code	DMSCBD	Contains directory information for compiler phases (part of DMSCOB after installation).
DMSCOB	COBOL	DMSCOB COBHAND	Initialization; link to compiler. Write ENTRY card for object module.

Figure 88. Load Module Directory

Chart VM (Part 1 of 3). DMSCOB (COBOL-CMS Interface Routine)

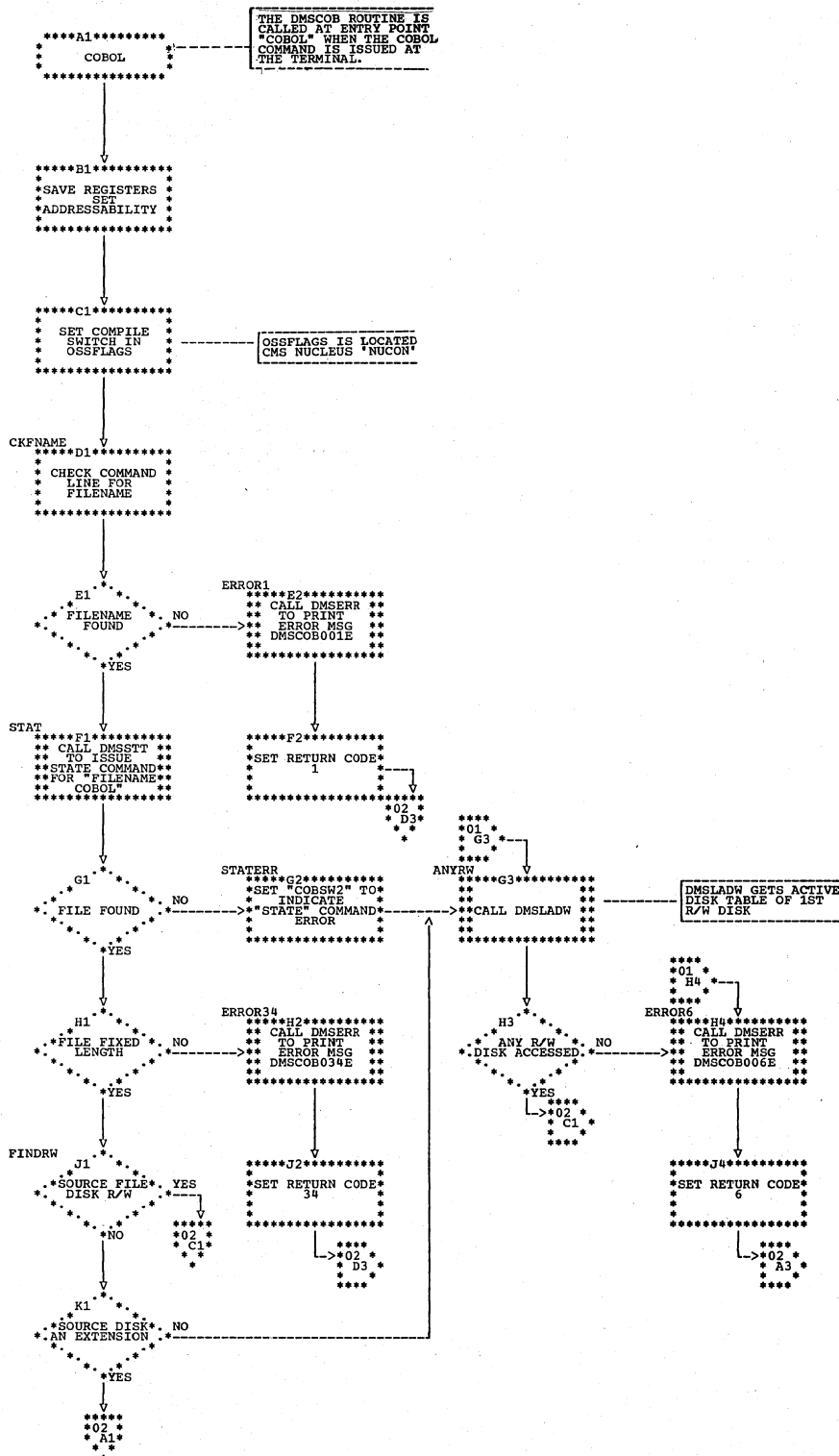
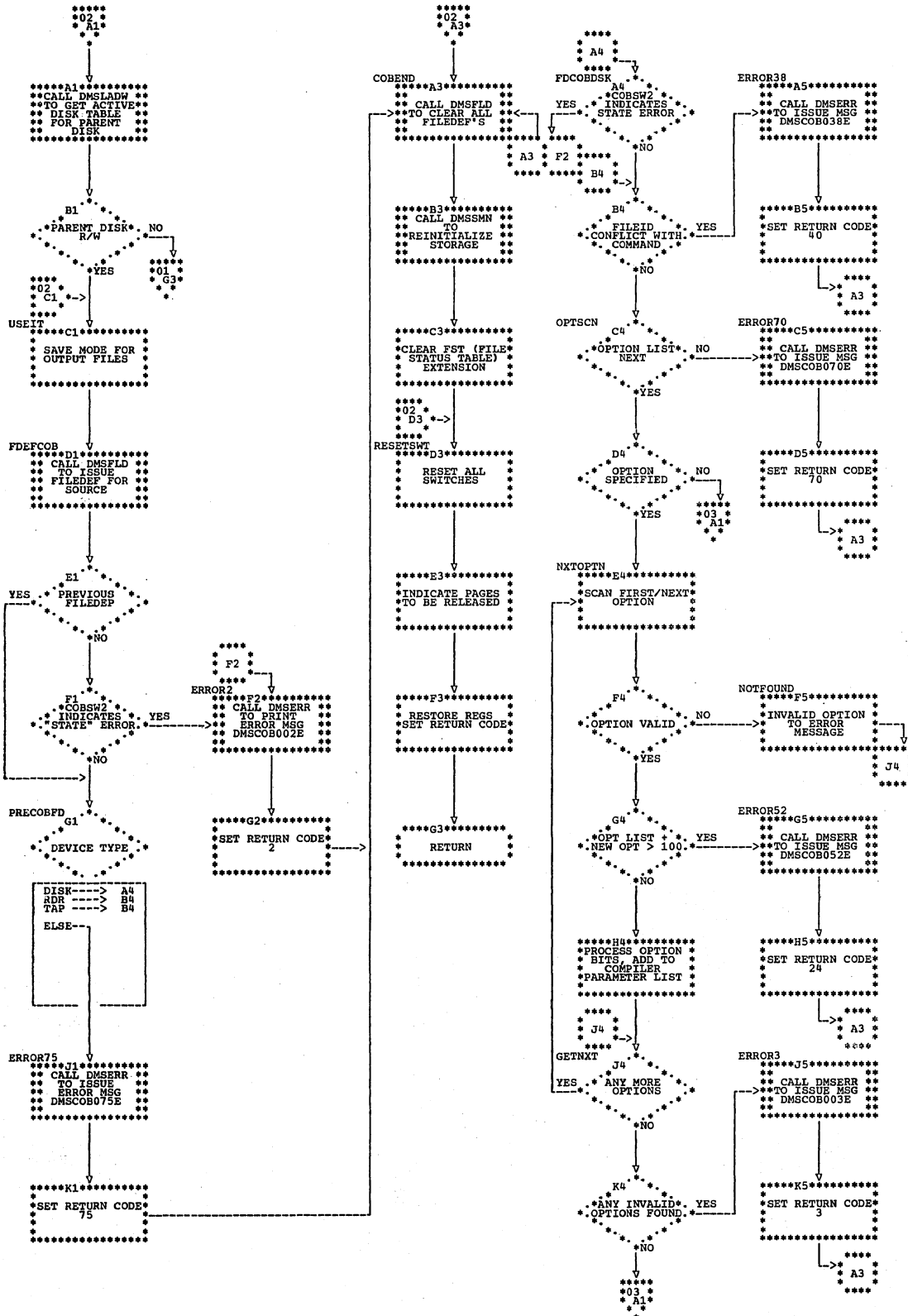




Chart VM (Part 2 of 3). DMSCOB (COBOL-CMS Interface Routine)





FLOWCHART LABEL DIRECTORY

Label	Chart	Definition		Reference	
		Page	Block	Page	Block
ANYRW	VM	01	G3	01	K1
				01	G2
				02	B1
				01	C1
				03	H2
				01	J4
CKFNAME	VM	01	D1	01	C1
				03	J2
				02	A3
				02	K1
				02	G2
				02	J2
COBCODE	VM	03	J2	03	H2
				01	J4
				02	K1
				02	G2
				02	J2
				02	B5
COBEND	VM	02	A3	02	D5
				02	H5
				02	K5
				03	K4
				03	J2
				03	B3
DOFINIS	VM	03	H4	03	D3
				03	F3
				03	H3
				03	E1
				01	E1
				03	J2
ERAS	VM	03	F1	03	E1
				01	E2
				03	E3
				03	G3
				03	J2
				02	F2
ERROR1	VM	01	E2	01	E1
				03	E3
				03	G3
				02	F2
				02	F1
				02	A4
ERROR12W	VM	03	E3	02	K4
				01	H2
				02	A5
				03	J2
				02	B4
				03	J2
ERROR16W	VM	03	G3	03	J2
				03	J2
				02	F2
				02	F1
				02	A4
				02	K4
ERROR2	VM	02	F2	02	F1
				02	A4
				02	K4
				01	H1
				01	H1
				02	H1
ERROR3	VM	02	J5	02	C4
				01	H2
				02	A5
				03	J2
				02	B4
				03	J2
ERROR34	VM	01	H2	01	H1
				02	A5
				03	J2
				02	G4
				03	D1
				02	C4
ERROR38	VM	02	A5	02	H1
				03	J2
				03	J2
				03	G1
				03	F1
				03	J1
ERROR4W	VM	03	A3	03	E2
				03	E2
				03	D2
				01	H1
				01	H1
				02	H4
ERROR52	VM	02	G5	02	G4
				03	D1
				03	D1
				02	C4
				02	H1
				02	H1
ERROR6	VM	01	H4	03	J2
				03	J2
				02	G1
				02	C1
				03	G1
				03	B2
ERROR70	VM	02	C5	03	F1
				03	F1
				03	J1
				03	E2
				03	E2
				03	D2
ERROR75	VM	02	J1	02	H1
				03	C3
				02	A4
				02	G1
				02	C1
				03	G1
ERROR8W	VM	03	C3	03	J2
				02	G1
				02	C1
				03	G1
				03	B2
				03	F1
FDCOBDSK	VM	02	A4	03	J1
				03	F2
				03	E2
				03	D2
				01	H1
				02	H4
FDEFDOB	VM	02	D1	02	F5
				02	D4
				02	K4
				02	F4
				02	D4
				02	D4
FDEFLLST	VM	03	H1	02	J4
				02	B4
				02	E1
				02	F1
				02	F1
				02	F1
FDEFLOAD	VM	03	C2	02	F4
				02	D4
				02	D4
				02	J4
				02	B4
				02	B4
FDEFTEXT	VM	03	G1	03	E1
				02	G1
				02	E1
				02	F1
				02	F1
				02	F1
FDEFUT1	VM	03	A2	02	F1
				02	F1
				02	F1
				02	F1
				02	F1
				02	F1
FDEFUT5	VM	03	E2	02	F1
				02	F1
				02	F1
				02	F1
				02	F1
				02	F1
FINDRW	VM	01	J1	01	H1
				02	H4
				02	F5
				02	D4
				02	F4
				02	D4
GETNXT	VM	02	J4	02	H4
				02	F5
				02	D4
				02	F4
				02	D4
				02	D4
NOOPT	VM	03	A1	02	F4
				02	D4
				02	F4
				02	D4
				02	D4
				02	D4
NOTFOUND	VM	02	F5	02	F4
				02	D4
				02	F4
				02	D4
				02	D4
				02	D4
NXTOPTN	VM	02	E4	02	F4
				02	D4
				02	F4
				02	D4
				02	D4
				02	D4
OPTSCN	VM	02	C4	02	F4
				02	D4
				02	F4
				02	D4
				02	D4
				02	D4
PRECOBFD	VM	02	G1	02	F4
				02	D4
				02	F4
				02	D4
				02	D4
				02	D4
RESETSWT	VM	02	D3	01	F2
				01	J2
				01	J2
				02	C3
				02	C3
				02	C3

Figure 90 (Part 1 of 2). Flowchart Label Directory

Label	Chart	Definition		Reference	
		Page	Block	Page	Block
STAT	VM	01	F1	01	E1
STATERR	VM	01	G2	01	G1
STDPLIST	VM	03	G2	03	F2
TERMFIL	VM	03	J1	03	H1
UPDTPDIR	VM	03	K4	03	J4
USEIT	VM	02	C1	01	J1
WRITE	VM	03	C5	01	H3
				02	B1
				03	B5

Figure 90 (Part 2 of 2). Flowchart Label Directory

DIAGNOSTIC AIDS

This section contains information for use in diagnosing difficulties with the COBOL-CMS Interface. Information is provided about:

- Data set activity.
- Register usage.
- Elements of program design.

DATA SET ACTIVITY

The DMSCOB routine issues FILEDEF commands for each of the data sets that are used by the compiler according to the user specified options. These commands are described in Figure 86 in the section "Operational Considerations."

REGISTER USAGE

Register usage is described in Figure 91.

ELEMENTS OF PROGRAM DESIGN

Elements in the design of the DMSCOB routine can be used to determine information in case of error. This section provides information about:

- Error messages issued by DMSCOB
- CMS service routines called by DMSCOB
- Register saving

Error Messages Issued by DMSCOB

The error messages issued by DMSCOB are listed in Figure 92.

Register	Usage
1-10	Work
11-12	Base
13	Work
14-15	Work/Linkage

Figure 91. Register Usage by DMSCOB

Message	Explanation	Issued by
DMSCOB004W	Minor errors were detected during compilation; successful execution of program is probable. Compilation is completed with error code of 4.	COBCODE ERROR4W
DMSCOB008W	Errors were detected during compilation; execution is possible. Compilation is complete with code of 8.	COBCODE ERROR8W
DMSCOB012W	Serious errors were detected during compilation. Successful execution of program is not probable. Compilation is completed with code of 12.	COBCODE ERROR12W
DMSCOB016W	Very serious errors were detected during compilation. Compilation is not complete. Results are not predictable. Error code of 16 is returned.	COBCODE ERROR16W
DMSCOB001E	No filename was specified in the COBOL command. Compilation is terminated with an error code of 24.	CKFNAME ERROR1
DMSCOB002E	The file named "filename COBOL" is not found on an access disk. Compilation is terminated with an error code of 28.	FDFPCOB ERROR2
DMSCOB003E	The option(s) specified is not valid for the COBOL command. Compilation is terminated with an error code of 24.	GETNEXT ERROR3
DMSCOB006E	No disk is currently accessed in a read/write status. Compilation is terminated with an error code of 36.	STATERR ERROR6
DMSCOB034E	The specified file must have fixed-length records to be acceptable as input to the CMS COBOL command. Compilation is terminated with an error code of 32.	STAT ERROR34
DMSCOB038E	A previously issued FILEDEF for DDname COBOL to a disk device did not contain the same filename and/or filetype as specified and implied by the COBOL command. Compilation is terminated with an error code of 40.	FDCOBDSK ERROR38
DMSCOB052E	More than 100 characters of options including one blank between each option were specified. Compilation is terminated with error code of 24.	NXTOPTN ERROR52
DMSCOB070E	The specified parameter is not expected in the COBOL command line. Compilation is terminated with error code of 24.	OPTSCN ERROR70
DMSCOB075E	The specified device type is illegal for input to the COBOL compiler. Compilation is terminated with an error code of 40.	PRECOBFD ERROR75

Figure 92. Error Messages Issued by DMSCOB

CMS Service Routines Called by DMSCOB

The DMSCOB routine calls several routines to perform functions such as erasing disk files, issuing FILEDEF commands and error messages. These routines and the subroutines of DMSCOB which call them are listed in Figure 93.

Register Saving

Normal OS/VS conventions are used for register saving. SAVE and RETURN macros are used in DMSCOB.

Service Routine Called	DMSCOB Subroutines
DMSAUPD	UPDTPDIR
DMSERR	ERROR1 ERROR12W ERROR16W ERROR2 ERROR3 ERROR34 ERROR38E ERROR4W ERROR52E ERROR6 ERROR70 ERROR75 ERROR8W
DMSERS	ERAS DOFINIS
DMSFLD	COBEND FDEFPCOB FDEFPLIST FDEFLOAD FDEFS FDEFTEXT FDEFUT1 FDEFUT5 TERMPIL
DMSFNSA	DOFINIS
DMSLADW	ANYRW FINDRW NOOPT
DMSBS	WRITE
DMSMN	COBEND NOOPT
DMSSTT	STAT

Figure 93. CMS Service Routines Called by DMSCOB

GLOSSARY

The words listed below are defined according to their usage in this publication. In the case of generic terms, the definitions would not necessarily be applicable outside of this context.

A-text (Assembler-text): Internal compiler text generated in phases 22, 21, 50 and 51, and used in phase 6 or 62, 63, and 64 to produce the object module for the linkage editor. This includes Data A-text, Procedure A-text, Procedure A1-text, and Optimization A-text. See "Section 5. Data Areas" for formats.

ACCESS: A group of routines loaded into main storage along with phases 1B, 20, 22, 21, 25, and 3 that build and access the compiler's dictionary. See "Appendix A: Table and Dictionary Handling."

ATF-text: An internal compiler text generated by phase 20 for phase 22. It is used in preparing entries for the dictionary. See "Section 5. Data Areas" for formats.

ATM-text: An internal compiler text generated by phase 4 for phase 45. It is used in creating P2-text for the UNSTRING verb. See "Section 5. Data Areas" for formats.

base locator (BL): A fullword cell in the TGT containing the address of a location in the data area of the object module. Phase 22 assigns one or more base locators to the Working-Storage Section, Communication Section, Report Section, and each file in the File Section. Phase 6 or 62 assigns a register to each base locator.

base locator for Linkage Section (BLL): Fullword in the TGT containing the address of an area passed as a result of an ENTRY statement, the address of a file label area provided by the control program, or the address of an SD area. BLLs are assigned by phase 22.

BL: See base locator.

BLL: See base locator for Linkage Section.

CD-text: A Data IC-text type that describes communication description entries.

COBOL library subroutines: Subroutines used for operations that are too extensive to be coded in-line each time they are used. Stored in the COBOL library and linkage edited with the object module to produce an executable load module.

COBOL space: An area in main storage representing the difference in length between the longest compiler phase and the phase currently processing, available to store tables and dictionary sections.

COMMON: A communications area resident in main storage throughout compilation as part of phase 00 and accessible via a DSECT to every phase. Used to store miscellaneous information and to pass information from one phase to another. The format is given in "Section 5. Data Areas".

CONTROL record: An 8-byte record associated with a DATA record. It is used during the sorting process when phase 6A is producing an alphabetized cross-reference listing.

COUNT Table: A part of the object module only when the COUNT option is specified. It contains entries for each procedure-name and verb in the source program.

critical program break: In Data IC-text, these are the Data Division header, Data Division section-names (File, Working-Storage, Linkage, Communication, and Report), the beginning and end of Q-Routine text, and the beginning and end of Report Writer text. In Procedure IC-text, they are Data Division headers, Report Section names, Procedure Division headers, Declaratives, End of Declaratives, beginning of debug packets, and end of program.

Data A-text: Text generated by phases 22 and 21 for phase 6 or 64 to generate the data and global table areas of the object module. See "Section 5. Data Areas" for format.

Data IC-text: Data Division information collected by phases 10 and 12 and merged with Environment Division information for use by phases 20, 22, and 21 in producing Data A-text and data-name dictionary entries.

**data operand:** A literal, figurative constant, or data item described by a Record Description entry (with a numeric level number) and used as an operand in the source program.

**DATA record:** A 48-byte record built by phase 6A which contains information about a procedure-name or a data-name obtained from input DEF-text. It may also contain references to the procedure-name or data-name.

**Debug-text:** Text generated by phase 6 or 63 and used by phase 65 for the STATE or SYMDMP option. It contains card numbers, their displacement within the object module, the priority of each segment, and discontinuity elements (produced by phase 63 only).

**DEF-text:** Text produced by phases 22 and 3 for phase 6A to use in generating the cross-reference listing, if the SXREF or the XREF option is in effect.

**delimiter:** An internal compiler text category that consists of the following elements: critical program breaks, verbs, source procedure-names at point of definition, and compiler-generated procedure-names at point of definition.

**delimiter pointer:** A field in dictionary entries for Data Division group items and Procedure Division section-names. For a group item, the delimiter pointer contains the section number and displacement of the next group item on the same or lower level. For section-names, it contains the section number and displacement of the next section-name.

**dictionary:** A special table, built by phases 1B, 22, and 21, into which all the attributes of every data operand, nondata operand, and procedure-name in the source program are collected. Unlike TAMER tables, the dictionary may be spilled onto external storage if storage space is not sufficient.

**dictionary attributes:** Descriptive information about every source program name placed into the dictionary by phase 1B for procedure-names and by phases 22 and 21 for Data Division names, and incorporated by phase 3 into P1-text.

**dictionary pointer:** The dictionary section number and displacement of a dictionary entry. The pointer is stored in the HASH table at a location depending on the hash value of the name the entry describes. If two or more names hash to the same value, special processing is required. See "Appendix A: Table and Dictionary Handling."

**E-text (Error-text):** Text generated by phases 10 through 51 whenever a source program error is encountered. The text is collected by phase 6 or 64 and used by phase 70 to generate error messages.

**element (text element):** One logical unit of a string of text, such as the description of a single data item or verb, preceded by a unique code identifying the element type. Each type has a fixed format, as given in "Section 5. Data Areas".

**entry (table entry):** Contiguously-placed information in a table that describes one item. All entries in any one table are usually fixed in length and format. See "Section 5. Data Areas".

**error:** A deviation from source language rules discovered in the source program.

**ESD-text:** External Symbol Dictionary cards that are punched in phase 6 or in phases 62 and 64, containing control information which identifies each external symbol in the module. They are used by the linkage editor to put the modules identified by the external-names into the load module.

**FD-text:** A type of Data IC-text that describes files.

**fragment:** A portion of code having a maximum size of one less than 64K bytes (65,535). A fragment begins with the first byte of a verb and ends with the last byte of a verb preceding the verb with a final relative displacement greater than 64K bytes. This unit is used in processing for the STATE or SYMDMP option.

**global table:** See Task Global Table and Program Global Table.

**GN:** See procedure-name, compiler-generated.

**hierarchy of operators:** The order in which operations must be performed in an arithmetic expression.

**IC-text (Internal Compiler text):** Text generated in phases 10, 12, and 1B from the source program, modified by subsequent phases, and eventually converted to A-text or dictionary entries. See "Section 5. Data Areas" for formats.

inline procedure: The set of procedural instructions that are part of the main sequential and controlling flow of the source program, that is, not part of the Declaratives Section or Sort input/output procedures.

Intermediate A-text: Text generated by phase 50 consisting of intermediate forms of Procedure A-text and Optimization A-text. Used only for input to phase 51.

Intermediate E-text: Text generated by phase 50 consisting of E-text to which an identification prefix has been added for phase 51.

intermediate result: Where the source program specified an arithmetic computation using more than two operands, the output of one step in the computation which is then used as an operand in the next step. At execution time, intermediate results are held in registers or in the TEMP STORAGE field of the TGT.

LD-text: A Data IC-text type that describes level-numbered entries.

Main Free Area: Main storage permanently allocated for tables and/or dictionary sections. The Main Free Area is in that portion of main storage immediately higher than COBOL space.

major code: A 4-bit binary code identifying the different types of dictionary entries. All data entries start with 0, and all nondata entries start with 1.

Master of an OCCURS clause with the DEPENDING ON option: A data-name for a variable-length group item which does not itself contain an OCCURS clause with the DEPENDING ON option, but at least one of its subordinate items at the next level does contain such an OCCURS clause.

minor code: A 4-bit binary code identifying the different categories of LD entries.

nondata operand: A file description, communication description, sort description, report description, or condition-name used as an operand in the source program.

object hierarchy: The order of all group and elementary items defined within the second group item in a MOVE CORRESPONDING statement.

object module: The output of a single execution of the compiler, and the input to the linkage editor.

Optimization A-text: Text generated by phases 50 and 51 to be used in phase 6 or 62 to eliminate storage duplication for virtuals, literals, and procedure-names. See "Section 5. Data Areas" for format.

out-of-line procedure: A section in the Declaratives portion of the Procedure Division, preceded by a USE statement.

OVERFLOW record: A 16-byte record chained to a DATA record. It contains references to the procedure-name or data-name which would not fit in the DATA record.

PGT: See Program Global Table.

phase: One load module of the compiler.

PN: See procedure-name, source program.

priority: The number assigned to a section in the Procedure Division. All sections having the same priority are loaded together as a segment. One of these, the root segment, resides in main storage throughout execution of the program. The other segments are loaded in order of the priority number, each segment overlaying the one before.

procedure-name, compiler-generated (GN): A point in the procedure instructions which the compiler designates to be the object of a branch instruction and, therefore, defines by means of a GN number.

procedure-name, source program (PN): A user-assigned name which appears in Area A of the source program Procedure Division once, and which may be used as the object of procedure-branching source statements.

procedure-name, variable (VN): The object of a branching instruction which may vary at execution time because it is modified by a PERFORM or ALTER statement.

Procedure A-text: A text of assembler language-like instructions produced by phase 51 and used by phase 6 in generating machine instructions for the object module or used by phases 62 and 63 in generating Procedure A1-text for phase 64. See "Section 5. Data Areas" for format.



Procedure A1-text: A text of assembler language-like instructions produced by phase 63 and used by phase 64 in generating machine instructions for the object module. See "Section 5. Data Areas" for format.

Procedure block: Unit of addressability in the optimizer version of the machine language program. Each Procedure block consists of approximately 4096 bytes of code. Most PNs and GNs within a Procedure block are addressed as displacements added to a base register which contains the address of the first instruction of the Procedure block.

Procedure IC-text: A text based on the source program Procedure Division and Report Writer statements in the Data Division. It is generated in its initial (P0) form by phases 12, 1B, and 22, and modified by phases 3 and 4. Phases 50 and 51 use it to generate Procedure A-text.

Program Global Table (PGT): An area of main storage in the object module which contains virtuals, literals, and address constants used by the object code of the generated program.

Progress message: A message written to the SYSTEM data set by phase 00 when the compiler is operating under the Time Sharing Option (TSO) of the operating system.

P0-text: The form of Procedure IC-text created first, by phases 12 (for Report Writer), 1B (for the Procedure Division), and 22 (for Q-Routines), as input to phase 3.

P1-text: The form of Procedure IC-text created by phase 3 as input to phase 4.

P2-text: The form of Procedure IC-text created by phase 4 as input to phases 50 and 51.

Q-Routines: Object module subroutines generated by phase 22 and used to calculate the length of a variable-length field defined by an OCCURS...DEPENDING ON clause and the location of the variably-located field which follows it.

qualified name: A user-assigned name which must be referred to along with another name to be unique, since it is used to define more than one item in the source program. For data-names, the qualifier is the name of a group to which this item is subordinate, and for procedure-names the qualifier is the section-name of the section in which the procedure is defined.

RFP-text: A text produced by phase 6 or 64 for phase 5A, if the SXREF or the XREF compiler option is in effect, to be used in producing a cross-reference listing. Each element indicates the card number of the statement in which a user-assigned name was referred to.

RIP-text: Information produced by phase 6 or 64 for the linkage editor, indicating all address constants in the object module which must be relocated.

root segment: That portion of a segmented object module which is resident in storage throughout object program execution.

SBL: See secondary base locator.

SD-text: The form of Data IC-text describing sort description entries.

secondary base locator (SBL): A fullword cell in the TGT containing the address of a field in the data area which is variably located because it follows an OCCURS...DEPENDING ON field in the same record.

section: A series of source program procedure instructions grouped under the same section-name.

segment: That portion of a segmented object module which constitutes one load module. It consists of all the instructions in those sections with the same priority.

segmentation: The feature by which the source programmer can organize his object program into several load modules, or segments, which may overlay each other during execution.

source module: The source program input to a single execution of the compiler.

subject hierarchy: The order of all group and elementary items defined within the first group item to appear in a MOVE CORRESPONDING statement.

table (compiler table): A contiguous area in main storage containing information of a particular type. A table is composed of a variable number of entries of a fixed, usually identical, format.

table-locator: A field divided conceptually into two parts, the first of which identifies the block within a table and the second of which specifies the displacement within the block at which an entry may be found.

TAMER (Table Access Management Executive Routines): A group of routines resident throughout compilation in phase 00 and accessible to all phases, which get space for, build, and provide access to compiler tables.

TAMER space: Space in main storage occupied by TAMER tables. It follows (that is, is in higher storage) COBOL space.

Task Global Table (TGT): An area of main storage in the object module containing information and work areas for use by the object program. Its format is given in "Appendix B: Object Module."

temporary result: A field temporarily allocated in the TS portion of the TGT to hold the result of an arithmetic computation, where that result must be placed in several receiving fields and each receiving field may require conversion or truncation. The space is allocated by phase 50.

text: Information created and written on a work file by one phase for subsequent reading by another phase, representing a stage in the translation of the source module into an object module.

TGT: See Task Global Table.

TIB (Table Information Block): One of 36 eight-byte cells in COMMON used to provide information about TAMER tables. The TIBs

are numbered and can be reassigned after a table has been released. See "Appendix A: Table and Dictionary Handling."

transient area: A portion of main storage reserved during execution time to contain segments which are not permanently resident in main storage. It contains one such segment at a time and is large enough to hold the largest non-resident segment in the program.

variable-length field: A data item described by an OCCURS...DEPENDING ON clause, whose length depends on the value of the object of DEPENDING ON.

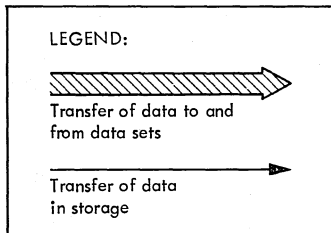
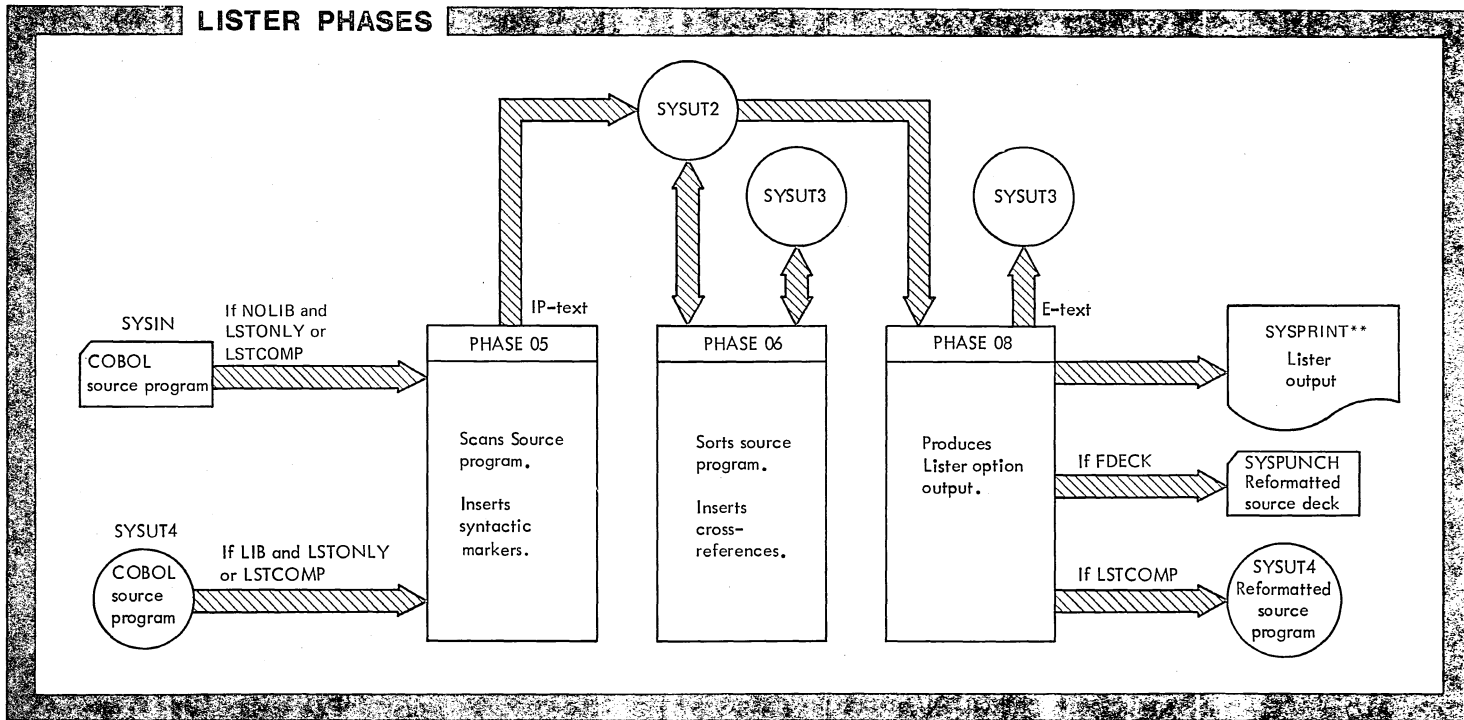
verb string: A verb and its operands.

virtual: The name of a procedure or table referenced by a procedure, but not defined in the source module. It is necessary because of a CALL to an external procedure or a branch to a COBOL library subroutine. At execution time, if neither the DYNAM nor the RESIDENT option was specified, the address of all procedures referred to by virtuals (which have been linkage edited into the load module) are stored in the Program Global Table. If the DYNAM or the RESIDENT option is in effect, library subroutines are dynamically loaded, and if the DYNAM option is in effect, subprograms are dynamically loaded.

DIAGRAMS



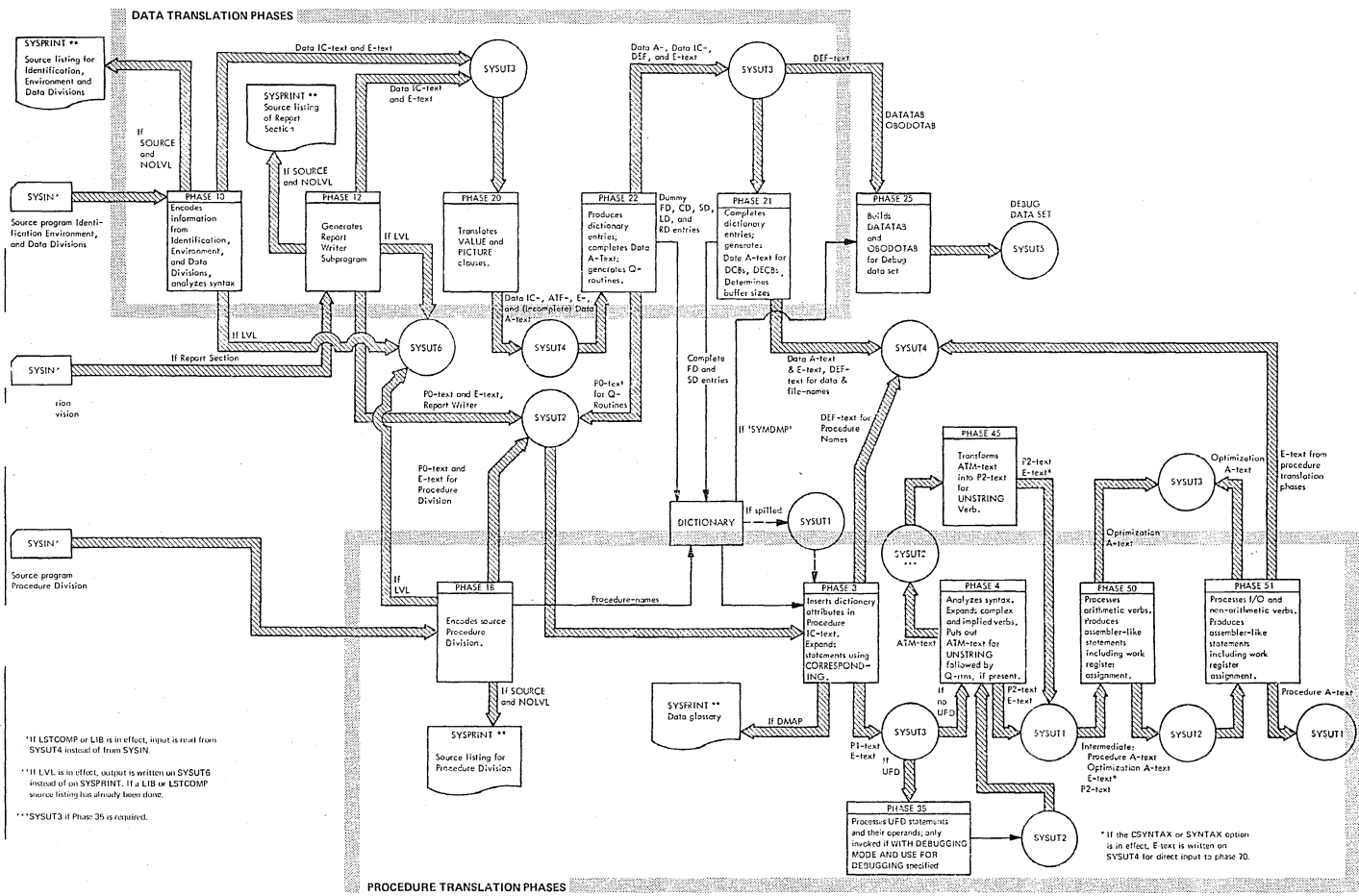
Diagram 1. Part 1. Design of the OS/VS COBOL Compiler



\*\*If LVL is in effect, output is written on SYSUT6 instead of on SYSPRINT.



Diagram 1. Part 2. Design of the OS/VS COBOL Compiler







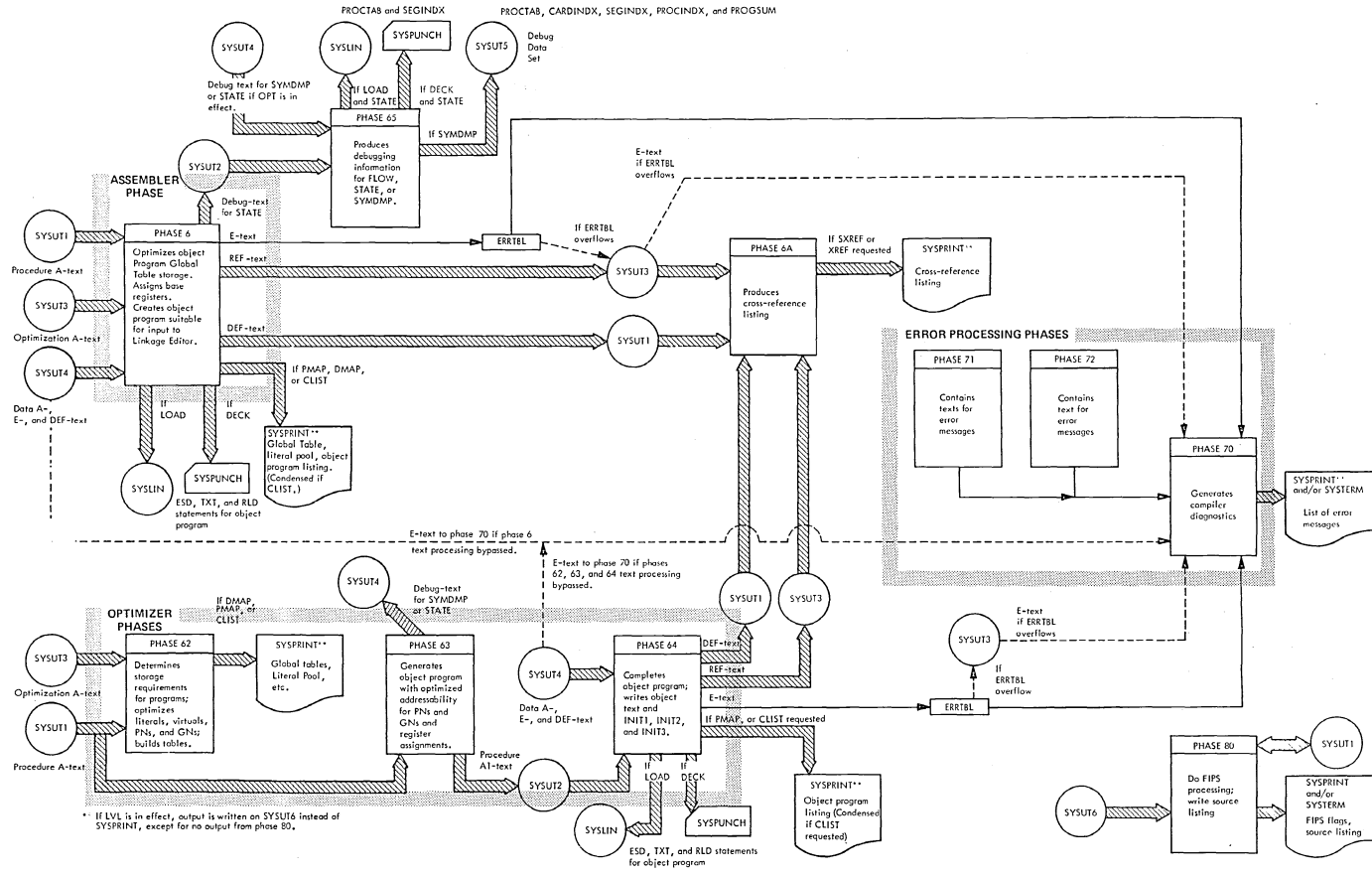


Diagram 1. Part 3. Design of the OS/VS COBOL compiler



Diagram 2. Part 1. Method of Operation: Table of Contents

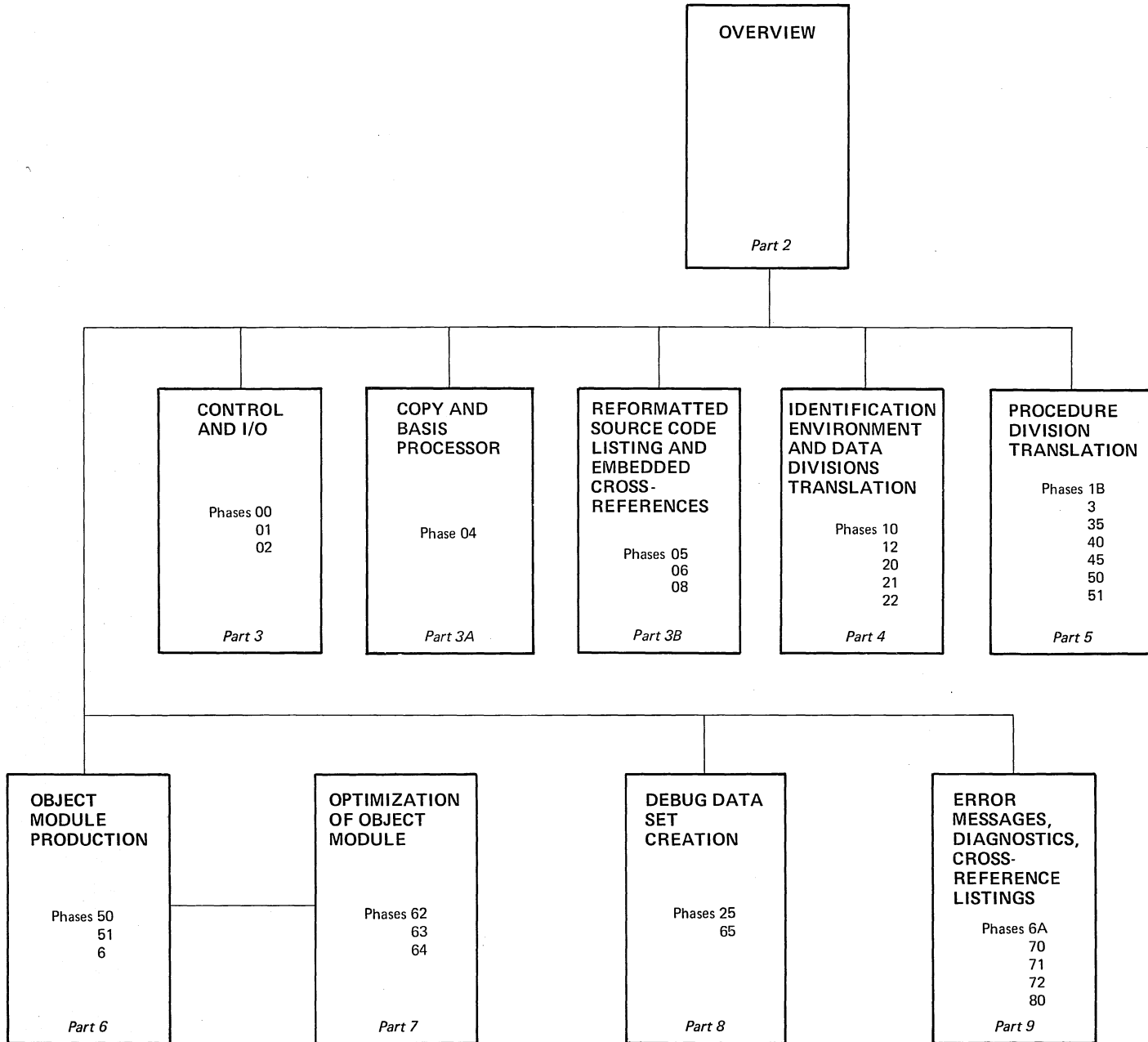




Diagram 2. Part 2. Method of Operation: Overview

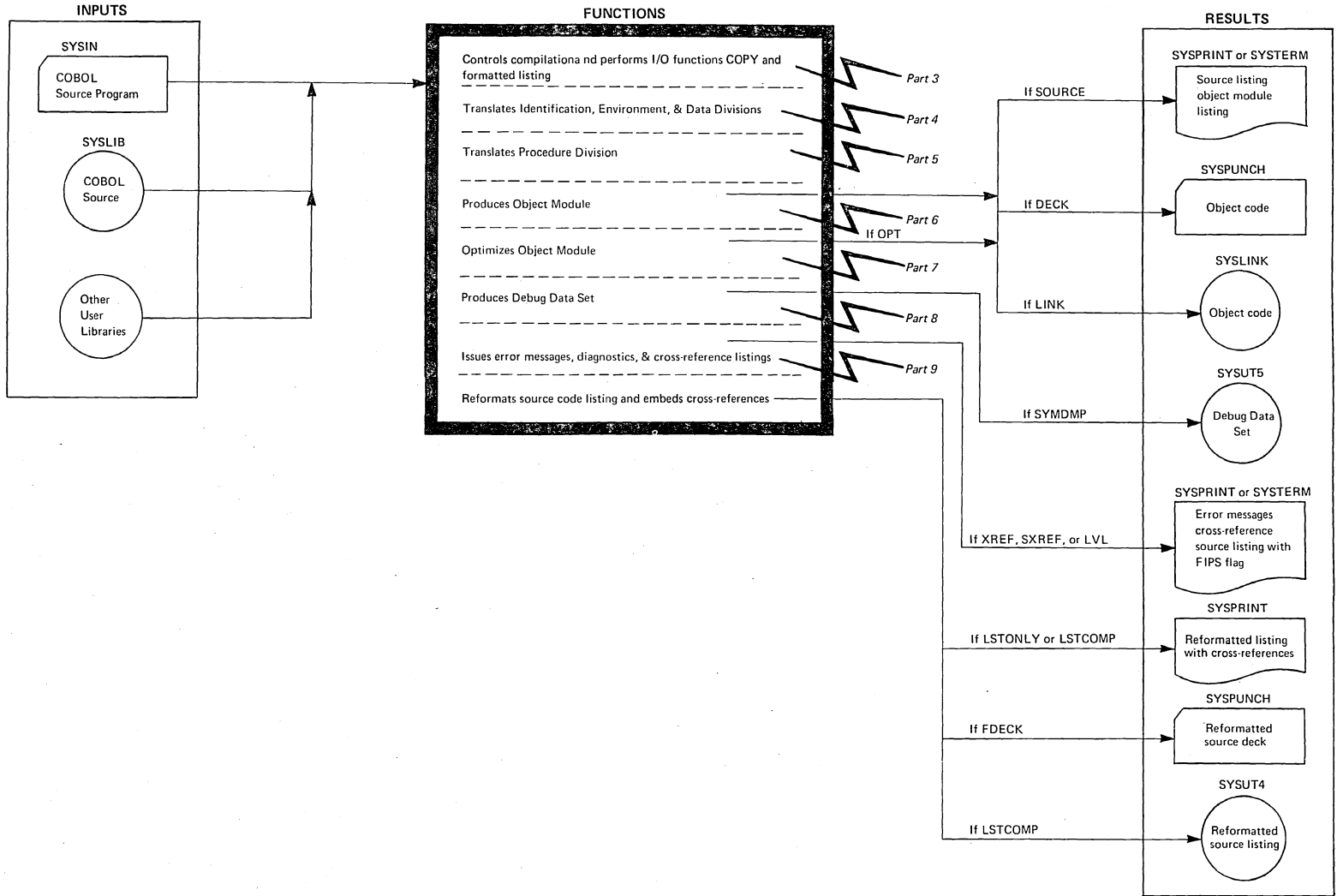
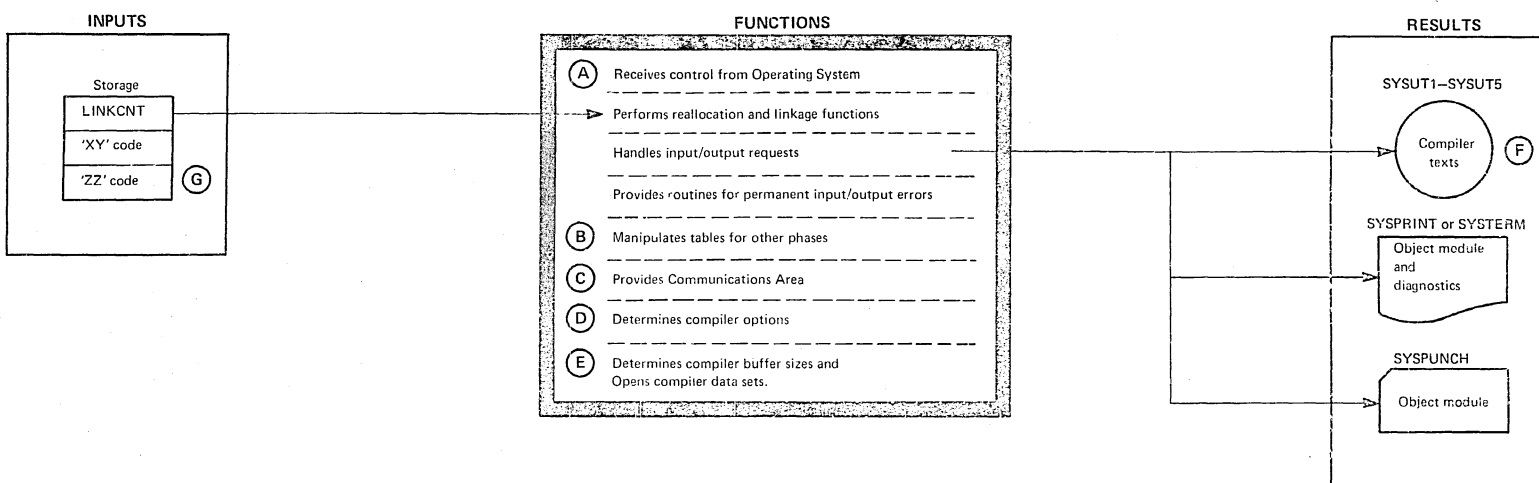




Diagram 2. Part 3. Method of Operation: Control and Input/Output

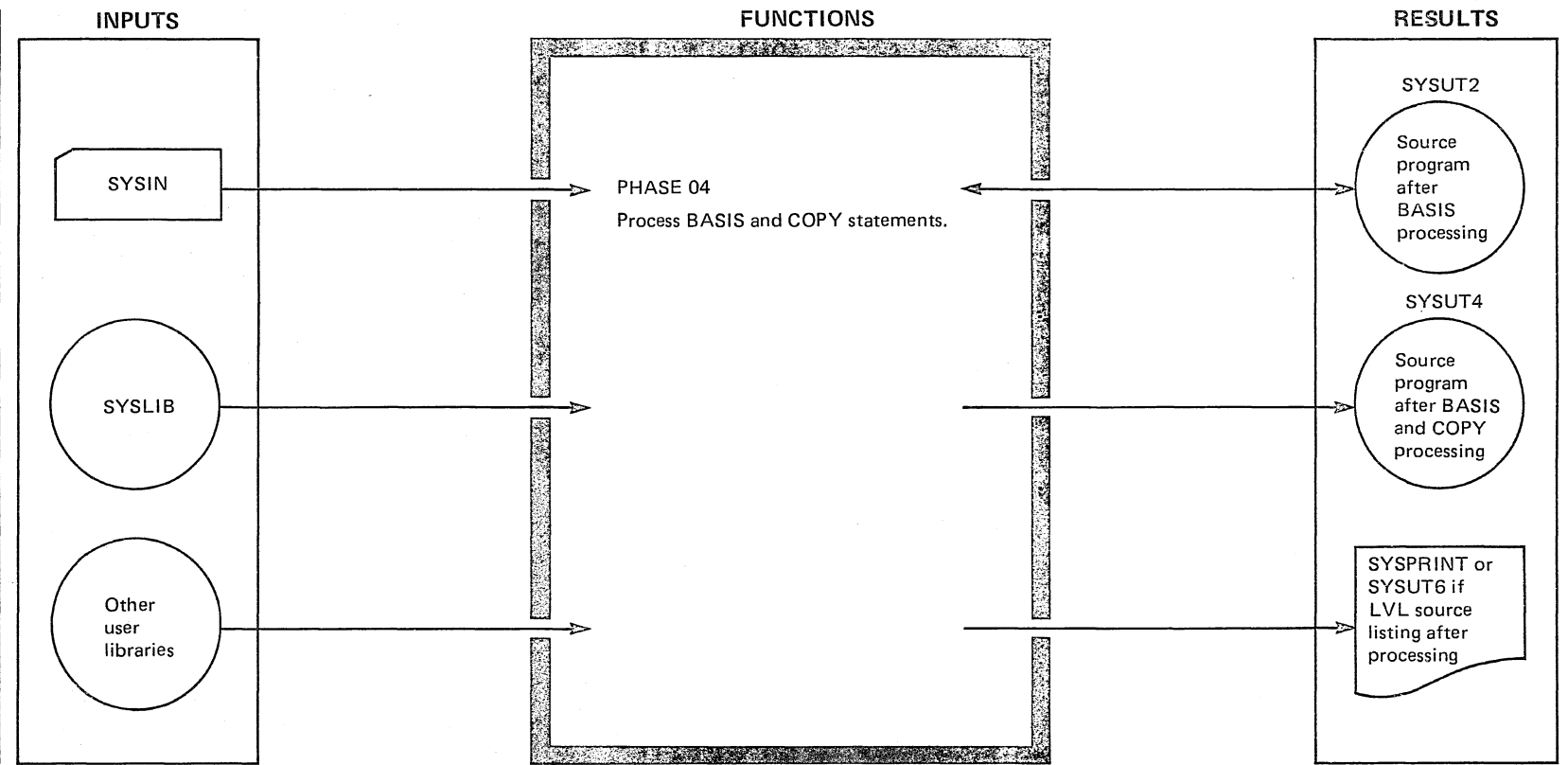


	Description	Module	Chapter	Chart
(A)	Receives control from; returns control to operating system.	IKFCBL00	Phase 00	AA
(B)	Provides Table Area Management Executive Routines (TAMER) for acquiring storage for and building tables.	IKFCBL00	Table and Dictionary Handling	
(C)	Provides Communications area (COMMON) used to pass information between phases.	IKFCBL00	Section 5: Communications Area (COMMON)	
(D)	Contains installation default values of compilation parameters, determines user options.	IKFCBL01	Phase 01	AB
(E)	Processes compilation parameters, determines buffer sizes for all phases, obtains storage for tables, dictionary, and buffers, enters information in COMMON, opens data sets.	IKFCBL02	Phase 02	AC
(F)	Activity of data sets and buffer assignments is summarized in Figure 10.			
(G)	'XY' and 'ZZ' codes are summarized in Figure 5.			





1 Diagram 2. Part 3a. COPY and BASIS PROCESSOR



Licensed Material - Property of IBM







Diagram 2. Part 4. Method of Operation: Identification, Environment, and Data Division Translation

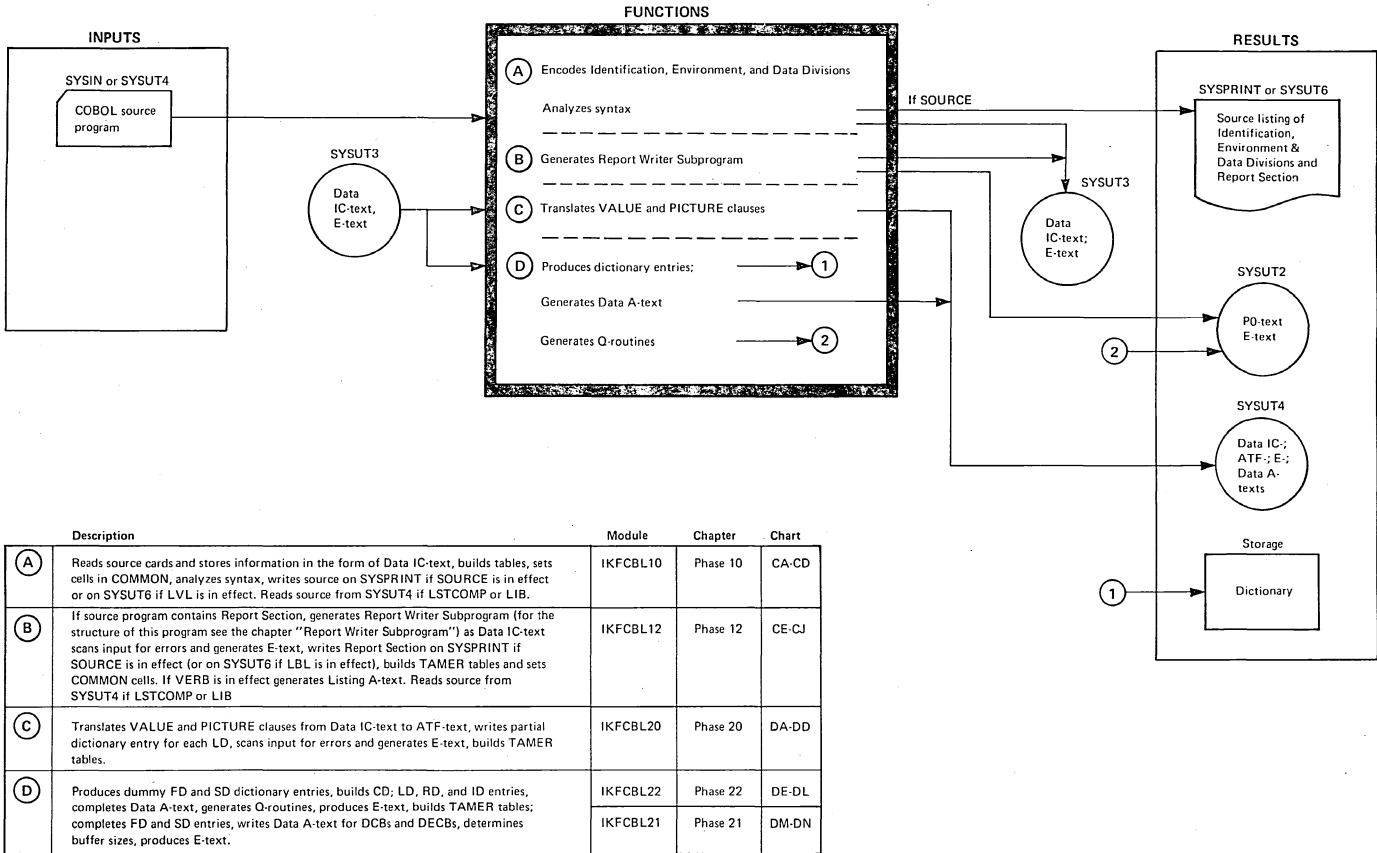
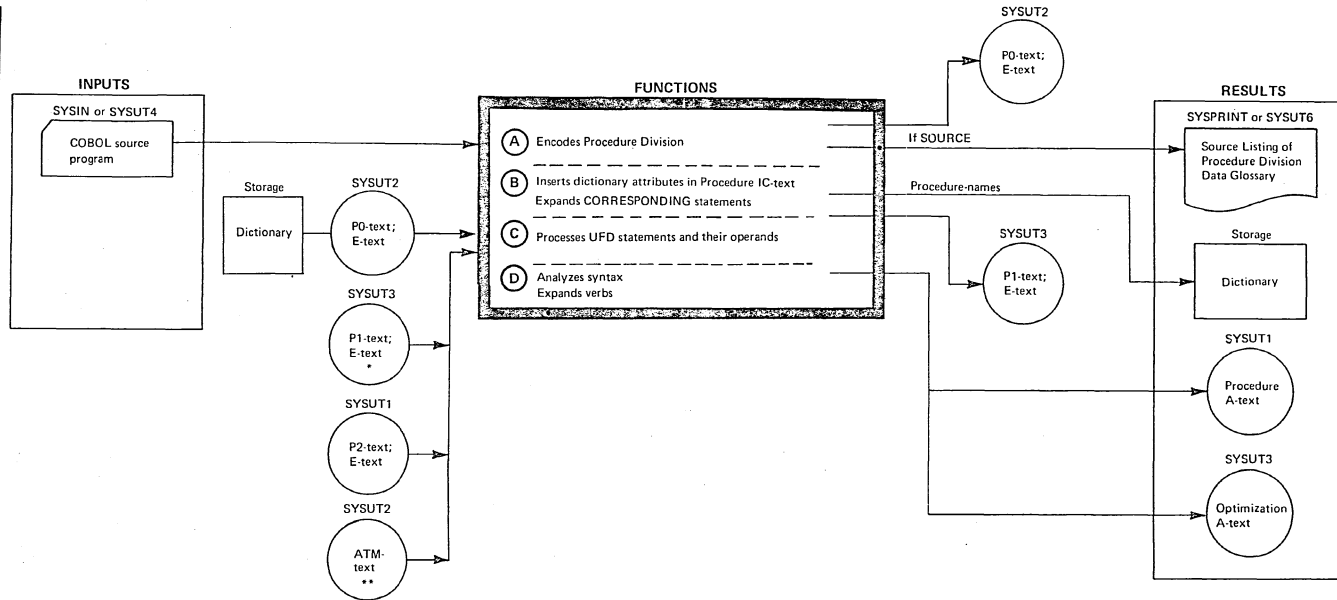




Diagram 2. Part 5. Method of Operation: Procedure Division Translation



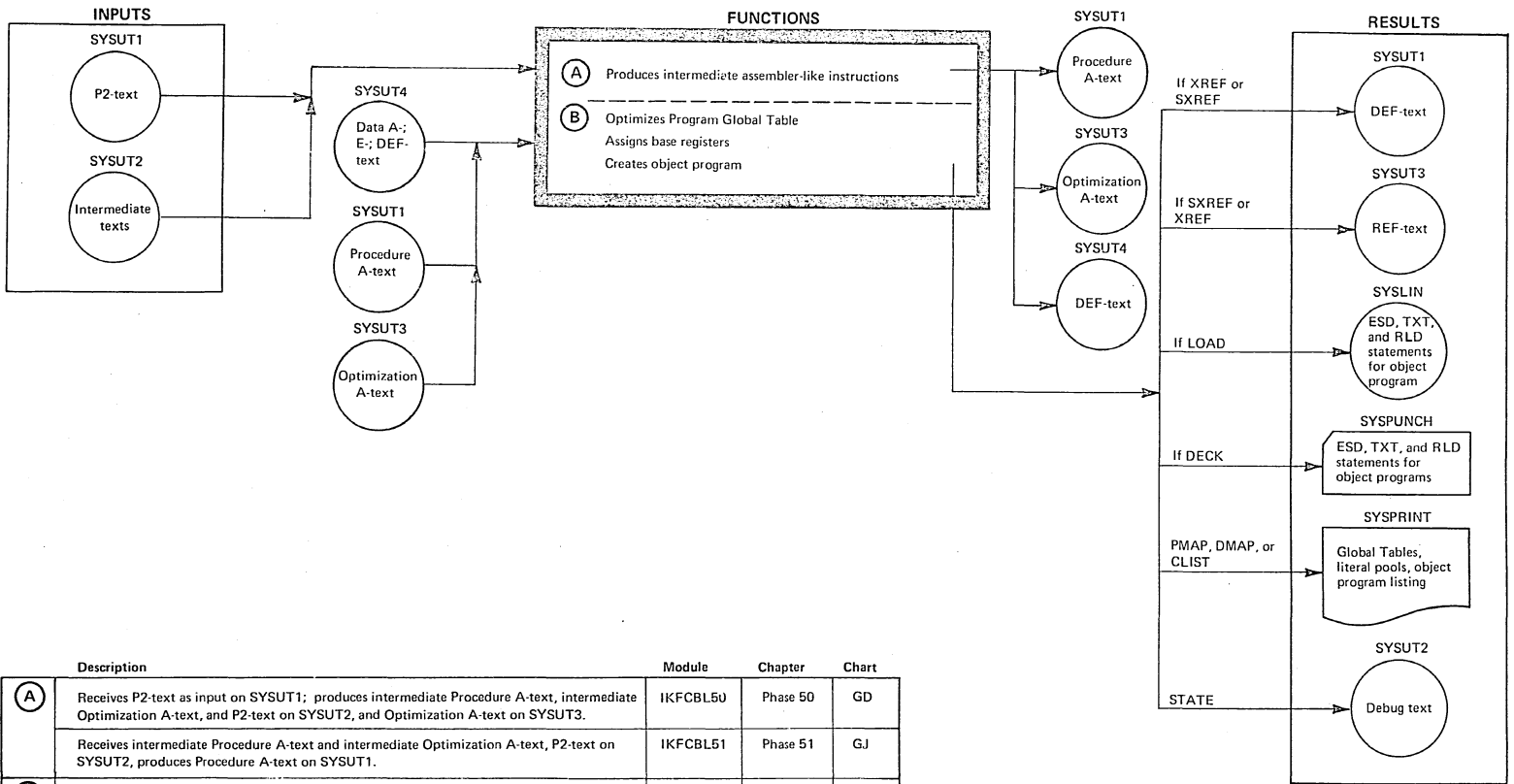
Description	Module	Chapter	Chart
(A) Reads Procedure Division of source program; creates dictionary entries for procedure-names; writes Procedure Division on SYSRINT if SOURCE is in effect (or on SYSUT6 if LVL is in effect); processes Declaratives Section; generates Listing A-text if VERB is in effect. Reads source programs from SYSIN or SYSUT4 if LIB or LSTCOMP.	IKFCBL1B	Phase 1B	CK
(B) Create P1-text; replaces source program names with dictionary attributes; builds Data Division Glossary of all source program data-names; performs special processing on procedure-names in segmented programs, verb strings, and verb strings with CORRESPONDING options; performs syntax analysis requiring dictionary; releases dictionary.	IKFCBL30	Phase 30	EA-EC
(C) Processes USE FOR DEBUGGING statements, adds to source programs the text necessary to cause invocation of the USE FOR DEBUGGING declaratives, only invoked if WITH DEBUGGING mode is specified and a USE FOR DEBUGGING is present.	IKFCBL35	Phase 35	ED-EI
(D) Translates P1-text from SYSUT3* to P2-text on SYSUT1; transforms P2-text to ATM-text on SYSUT2** for UNSTRING verb, expands complex and implied verbs.	IKFCBL40	Phase 40	FA-FC
Analyzes and translates UNSTRING verb from ATM-text on SYSUT2** to P2-text on SYSUT1.	IKFCBL45	Phase 45	FD-FF
Processes arithmetic verbs from SYSUT1 to SYSUT2.	IKFCBL50	Phase 50	GA-GE
Process input/output verbs and other non-arithmetic verbs from SYSUT2 to SYSUT1.	IKFCBL51	Phase 51	GF-GJ

\* SYSUT2 if Phase 35 has been invoked.  
 \*\* SYSUT3 if Phase 35 has been invoked.





Diagram 2. Part 6. Method of Operation: Object Module Production



	Description	Module	Chapter	Chart
A	Receives P2-text as input on SYSUT1; produces intermediate Procedure A-text, intermediate Optimization A-text, and P2-text on SYSUT2, and Optimization A-text on SYSUT3.	IKFCBL50	Phase 50	GD
	Receives intermediate Procedure A-text and intermediate Optimization A-text, P2-text on SYSUT2, produces Procedure A-text on SYSUT1.	IKFCBL51	Phase 51	GJ
B	When OPT is not in effect; determines object program storage for Task Global Table and Program Global Table; optimizes literals, virtuals, source program procedure-names, and compiler-generated procedure-names; generates and writes machine language instructions; writes object text for data area of program, writes object text for initialization routines; passes E-text to phase 70. When OPT is in effect, the functions described in Part 7 of this diagram take place instead.	IKFCBL60	Phase 6	HA-HF



Diagram 2. Part 7. Method of Operation: Optimization of Object Module (Optional)

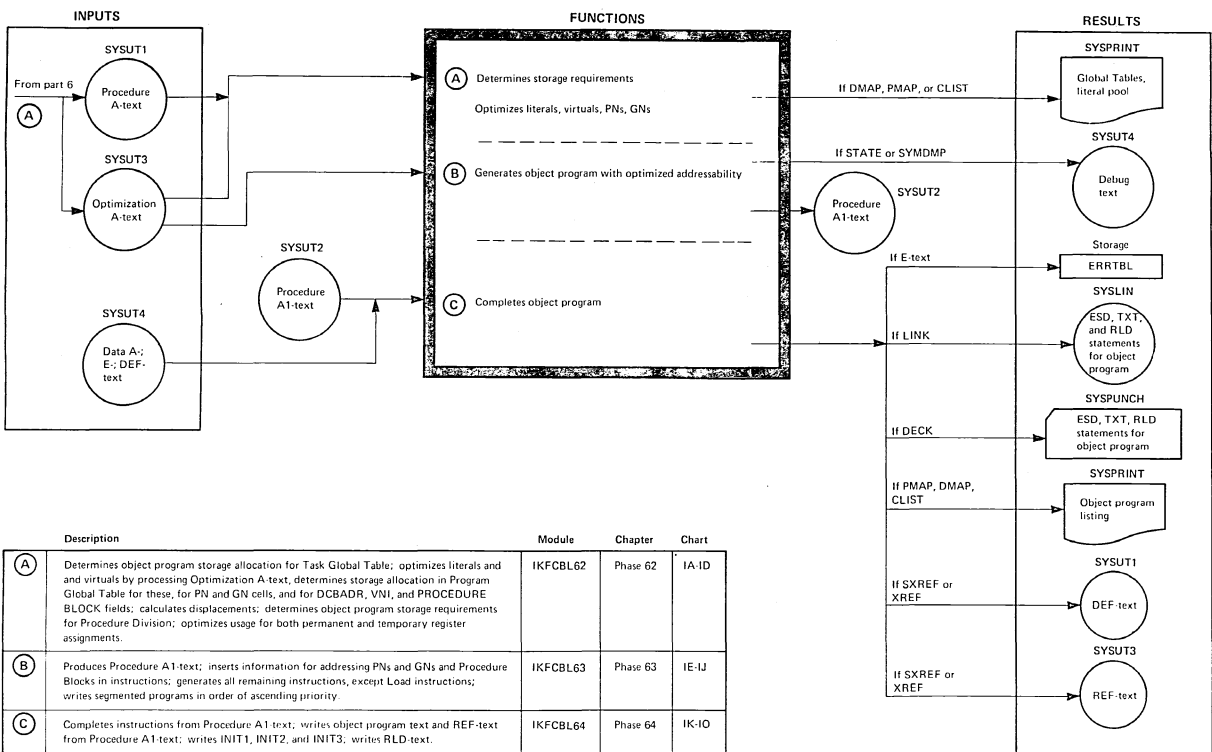
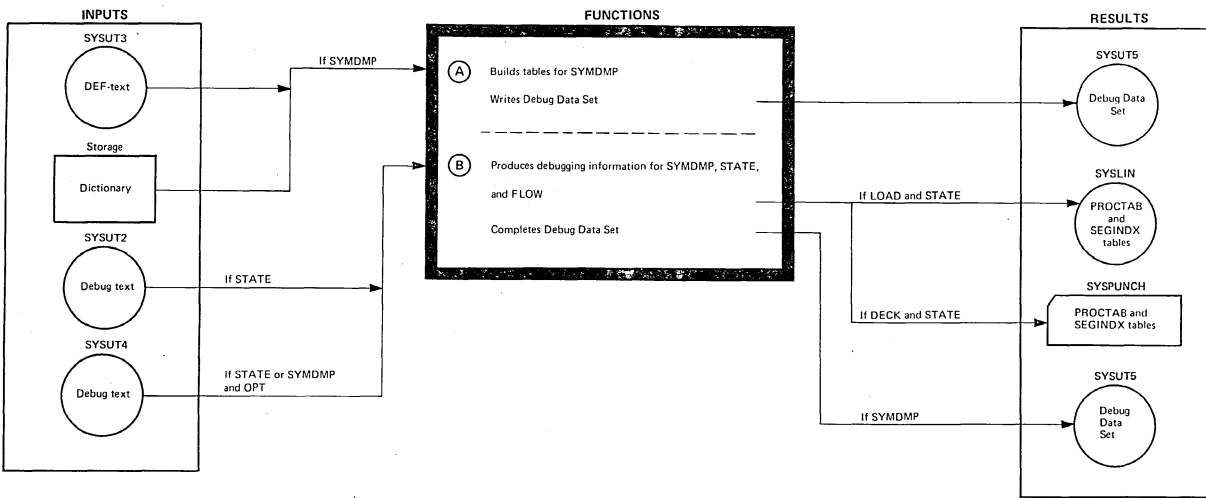




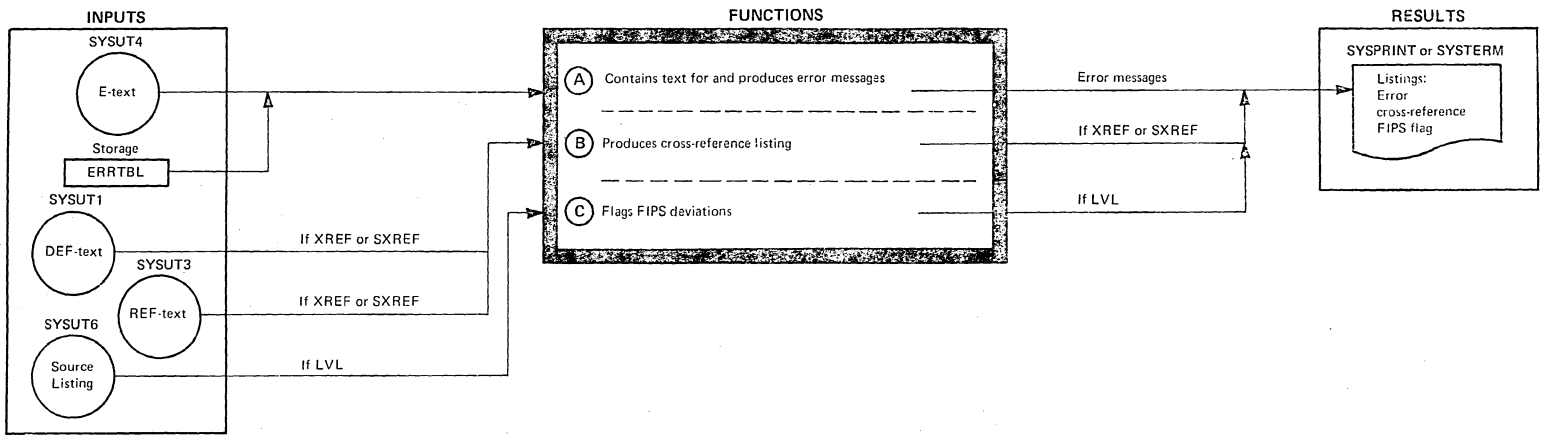
Diagram 2. Part 8. Method of Operation: Debug Data Set Creation (Optional)



	Description	Module	Chapter	Chart
(A)	If SYMDMP is in effect, builds OBODOTAB table for OCCURS... DEPENDING ON clauses; builds DATATAB table; writes tables on SYSUT5 data set.	IKFCBL25	Phase 25	DO-DR
(B)	For the FLOW option, stores number of traces requested in DEBUG TABLE of Task Global Table. For STATE, produces PROCTAB and SEGINDX tables and writes them in object module. For SYMDMP, produces CARDINDX, PROCINDX, PROGSUM, PROCTAB, and SEGINDX tables and writes them on Debug Data Set (SYSUT5).	IKFCBL65	Phase 65	IP-1Q



Diagram 2. Part 9. Method of Operation: Error Messages, Diagnostics, and Cross-Reference Listings



Description	Module	Chapter	Chart
<b>(A)</b> Produces E-text as it scans source program listing and analyzes syntax.  Contains text for error messages.  Formats messages and prints them on SYSPRINT or, if TERM is in effect, on SYSTEM.	IKFCBL04 IKFCBL10 IKFCBL12 IKFCBL18 IKFCBL20 IKFCBL22 IKFCBL21 IKFCBL25 IKFCBL30 IKFCBL35 IKFCBL40 IKFCBL45 IKFCBL50 IKFCBL51 IKFCBL60 IKFCBL62 IKFCBL63 IKFCBL64 IKFCBL65	Phase 04 Phase 10 Phase 12 Phase 18 Phase 20 Phase 22 Phase 21 Phase 25 Phase 3 Phase 35 Phase 4 Phase 45 Phase 50 Phase 51 Phase 6 Phase 62 Phase 63 Phase 64 Phase 65	
	IKFCBL71,70 IKFCBL72	Phases 70 71 72	
	IKFCBL70	Phases 70 71 72	JA
<b>(B)</b> If XREF is in effect, produces a source-ordered cross-reference listing; if SXREF is in effect, produces an alphabetically ordered cross-reference listing.	IKFCBL6A	Phase 6A	IR
<b>(C)</b> If LVL is in effect, scans COBOL source program after compilation is complete and produces messages indicating deviations from the Federal Information Processing Standard (FIPS).	IKFCBL80	Phase 80	MA

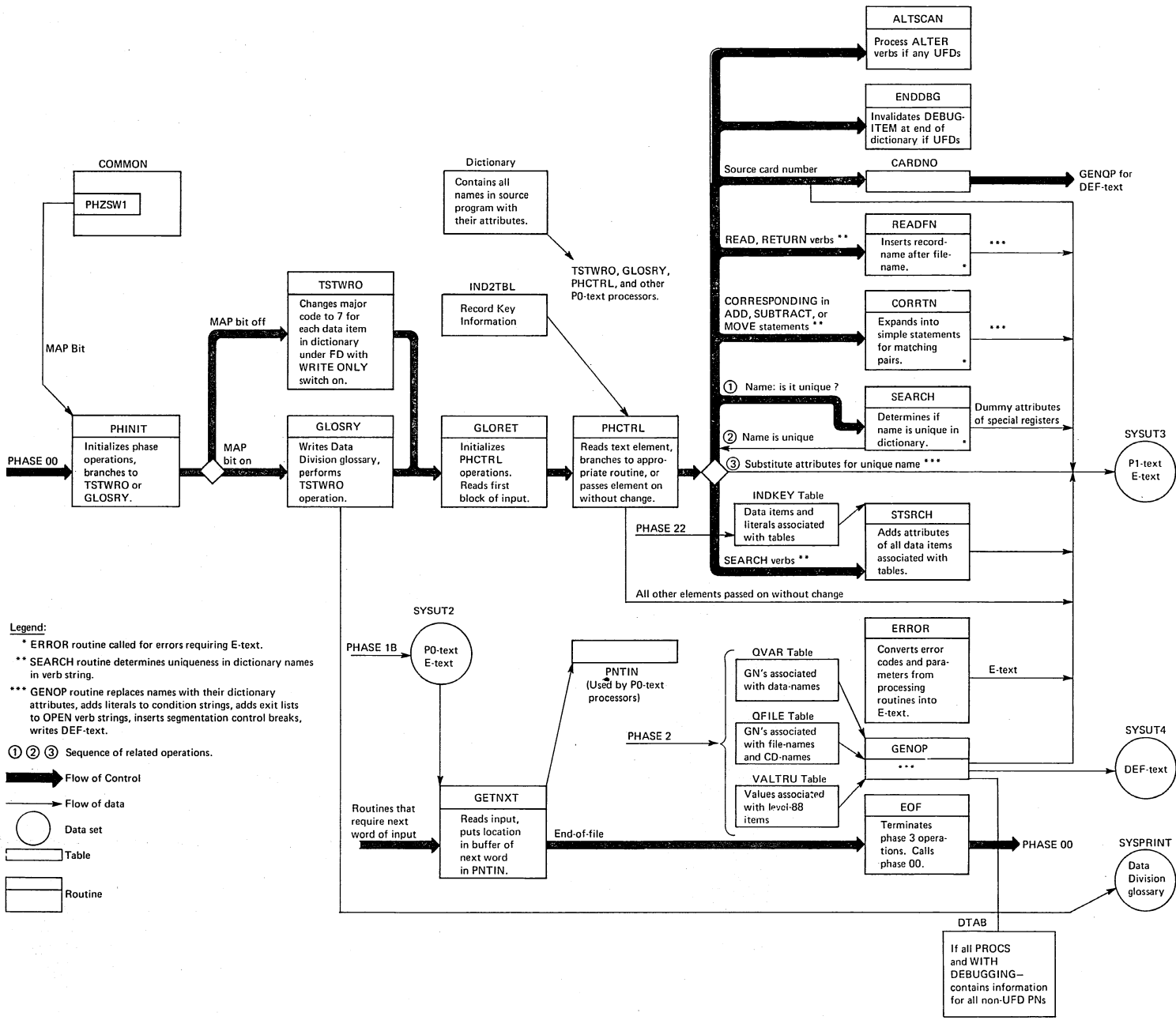








Diagram 4. Phase 3 Operations



- Legend:**
- \* ERROR routine called for errors requiring E-text.
  - \*\* SEARCH routine determines uniqueness in dictionary names in verb string.
  - \*\*\* GENOP routine replaces names with their dictionary attributes, adds literals to condition strings, adds exit lists to OPEN verb strings, inserts segmentation control breaks, writes DEF-text.
  - ① ② ③ Sequence of related operations.
  - ➔ Flow of Control
  - ➔ Flow of data
  - Data set
  - ▭ Table
  - ▭ Routine



Diagram 5. Phase 62 Operations

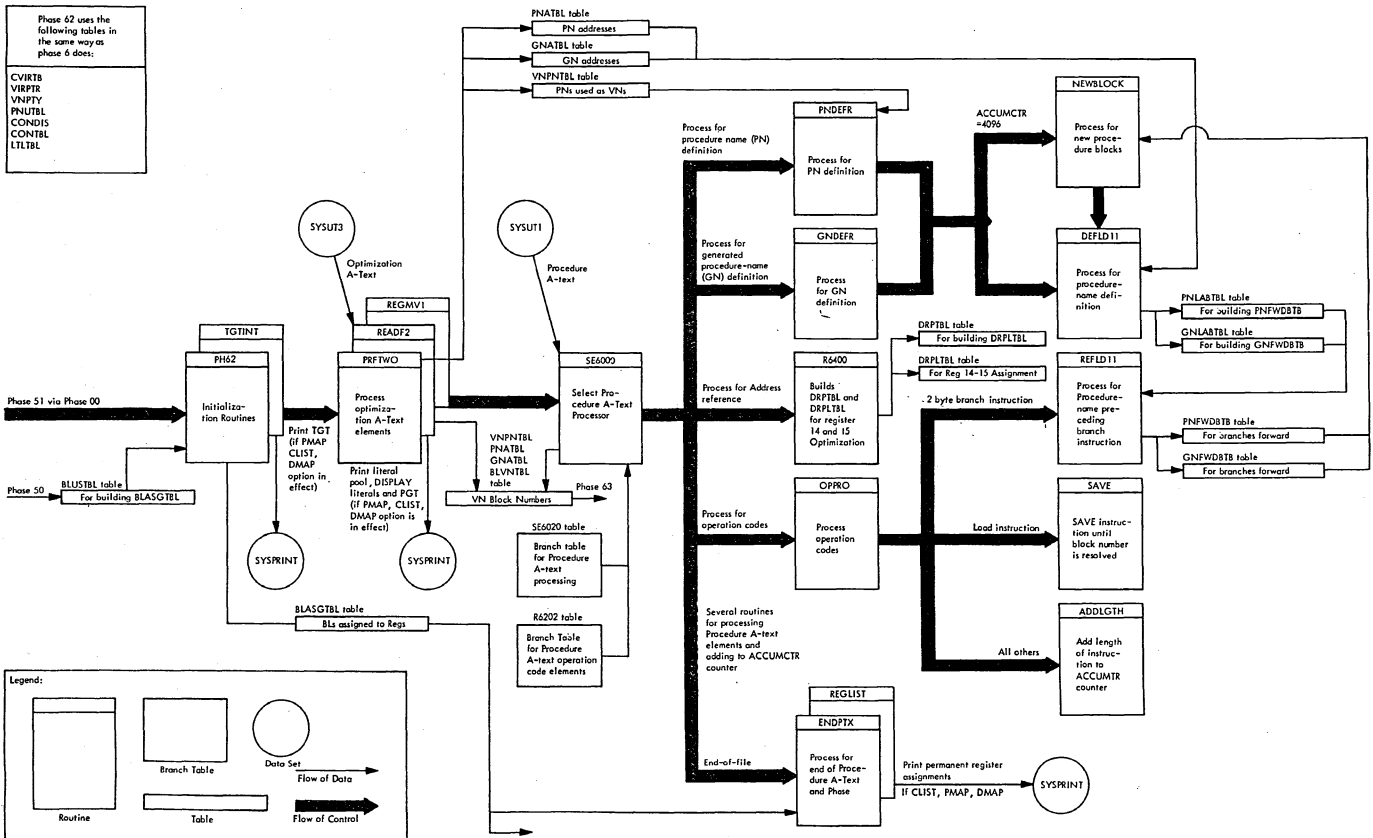




Diagram 6. Phase 63 Operations

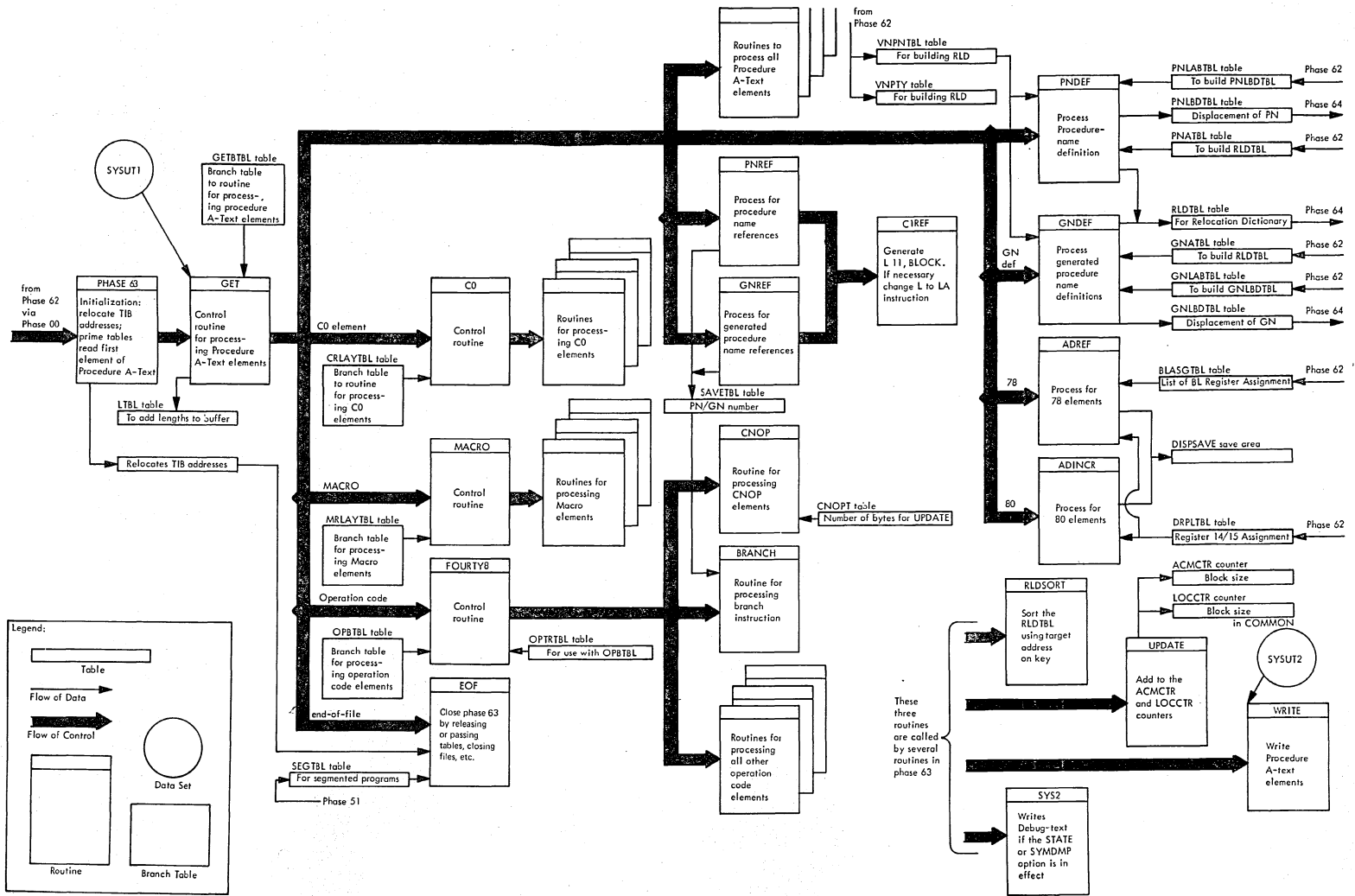
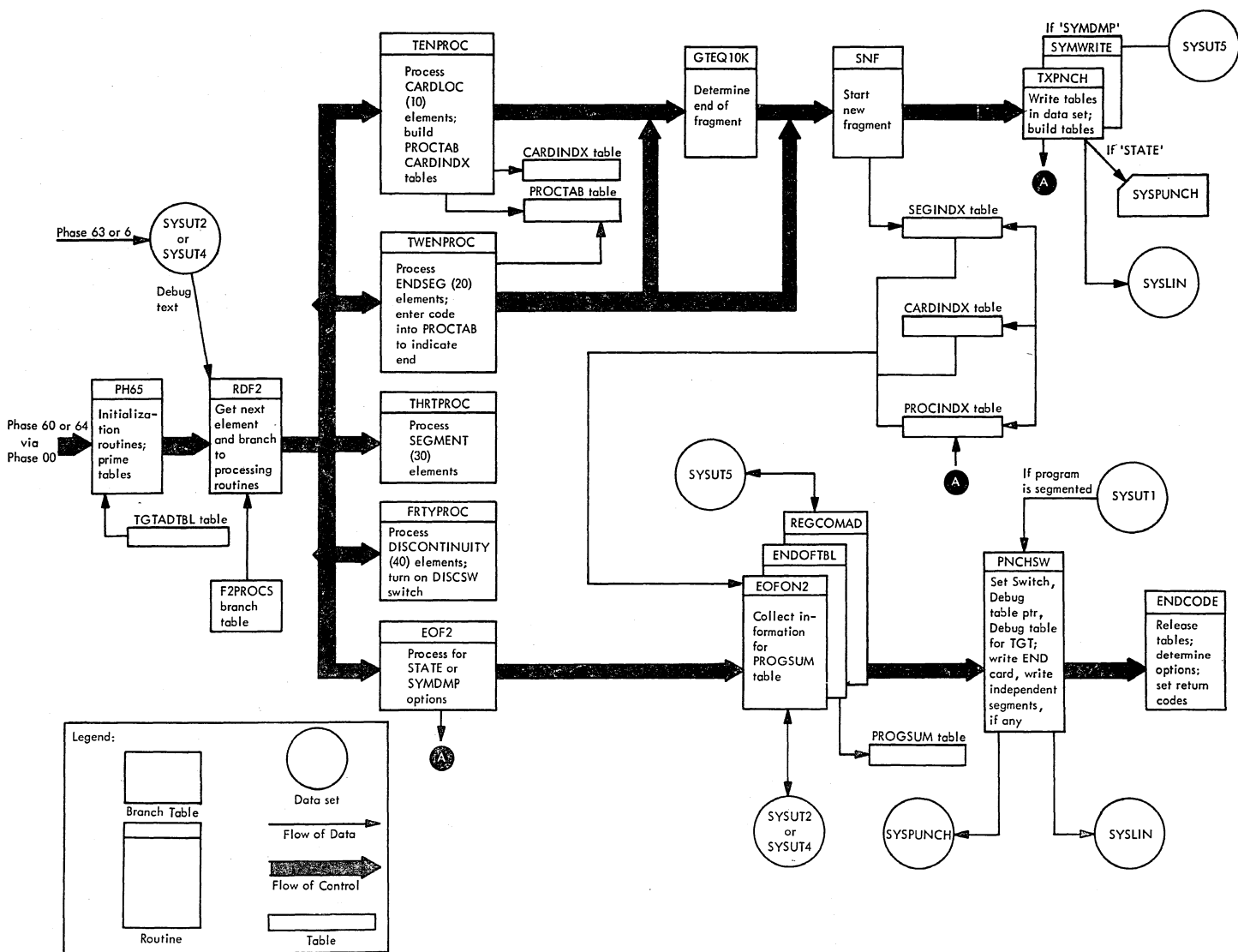






Diagram 7. Phase 65 Operations





INDEX

(Where more than one page reference is given, the major reference is first.)

- nnnn (Report Writer data-name)
  - description 522
  - phase 22 generation of 62
- A-text
  - (see also Listing A-text; Data A-text; Optimization A-text)
  - constant 127,132
  - definition 542
  - direct 127,132
  - generator routine
    - description 127-132
    - phase 50 flowchart 269
    - phase 51 flowchart 275
    - use in Arithmetic Translator routines 124
  - A-text generator
    - description 127-132
    - flowcharts
      - phase 50 269
      - phase 51 275
  - A(INIT1) field (TGT)
    - description 507
    - location 505
  - ABEND codes 483-484
  - abnormal termination
    - compiler processing 31,32
    - SYMDMP processing for 85
  - ABS.LIN (Report Writer data-name) 523
  - access methods
    - BDAM
      - address elements 80
      - block address elements 80
      - buffer areas 82,83
      - constant definition elements 80
    - BISAM
      - address elements 80
      - block address elements 80
      - buffer areas 82,83
      - constant definition elements 80
    - BSAM
      - address elements 80
      - block address elements 80
      - buffer areas 82,83
      - constant definition elements 80
    - phase 21 processing 79-84
    - QISAM
      - block address elements 80
      - buffer area size 82
      - buffer areas 82,83
    - QSAM
      - block address elements 80
      - buffer areas 83
      - buffer area size 82
      - Q-routine generation 83
    - VSAM
      - address element 80
      - buffer area size 83
  - ACCESS routines
    - definition 542
    - description 492-500
    - ENTDEL 494-495
    - ENTNAM 494
    - ENTPTR 494
    - GETPTR 494
    - LATACP 496
    - LATGRP 496
    - LATRNM 495
    - LATRPT 495
    - LDELNM 495-496
    - LOCNXT 495
    - use
      - phase 1B 67
      - phase 3 88
  - ACCESW cell (COMMON) 325
  - ACCMET subroutine 79
  - ACCUMCTR counter
    - incrementing of 171
    - use in phase 62 169
  - ACMCTR counter
    - use in ADINCR routine 176
    - use in phase 63 177
  - ADATAB cell (COMMON) 330
  - ADD string 125
  - ADDLGTH routine 583
  - ADDRCARD cell (COMMON)
    - description 333
    - use in phase 02 48
    - use in phase 1B 66
    - use in phase 10 56
  - addressing parameters 72
  - address calculation
    - indexed references 123
    - subscripted references (see subscripted references)
  - address constant definition element
    - Data A-text format 409
    - DCB 80,81
    - DECB 80,81
    - phase 21 processing 80,81
    - phase 6 processing 160
    - phase 64 action 182
  - address constants of TAMER routines 325
  - address increment elements
    - phase 63 processing of 176
    - phase 64 action 185
    - procedure A1-text format 441
    - use in phase 62 169
  - address increments
    - phase 6 processing 155
    - phase 63 processing 176
    - phase 64 processing 185
    - Procedure A-text format 435
  - address reference elements
    - phase 63 processing of 176
    - phase 64 action 185

address reference elements (continued)  
 procedure A1-text format 441  
 use in phase 62 169

address references  
 phase 6 processing 155  
 phase 64 processing 185  
 Procedure A-text format 435

ADINCR routine  
 description 176  
 diagram of 583  
 flowchart 291

ADREF routine (phase 63)  
 description 176  
 diagram of 583  
 flowchart 291  
 incremented address processing 176

ADREF routine (phase 64), flowchart 296

ADSTAT cell (COMMON) 325

ADV option  
 bit in COMMON 334  
 description 29

AGETALL cell (COMMON) 328

AHEADER cell (COMMON) 338

AINSRT cell (COMMON) 325

ALIBEOD cell (COMMON) 339

ALIBSYNA cell (COMMON) 339

ALL literal, Procedure IC-text format 421

allocating storage for the PGT,  
 phase 62 168-170

alphanumeric literal references, Procedure  
 IC-text  
 P0 format 411-418  
 P1A format 424  
 P1 format 419-423  
 P2 format 425-432

ALPHTBL table  
 format 341  
 use in phase 10 57

ALS-ROUT routine 64

ALSTAM cell (COMMON) 325

ALTER statement, OPT option  
 processing 105-109

ALTSCN routine 58,96

AMAINF cell (COMMON) 325

American National Standard Code for  
 Information Interchange (see ASCII)

AMICTR cell (COMMON) 330

AMILOC cell (COMMON) 331

AMOVDC cell (COMMON) 328

ANLZUFDS routine  
 description 101  
 flowchart 255

APLSCALL cell (COMMON) 331

APOST option  
 bit in COMMON 333  
 description 28

APPLY clause 58

APPLY WRITE-ONLY clause, phase 3 processing  
 for 88

APPWRO switch (COMMON), use in phase 3 88

APRIME cell (COMMON) 325

arithmetic operator, Procedure IC-text  
 P0 format 412  
 P1 format 422

arithmetic processing switches 125

arithmetic translator routines  
 description 124-127  
 switches for 125

arithmetic verb strings 124-127

assembler-text (see A-text)

ATF-text  
 definition 542  
 format 408  
 generation by phase 20 71-72

ATM-text  
 definition 542  
 description 433  
 format (same as P2) 425  
 input to phase 45 118  
 phase 4 processing 104

ATM-text analysis, phase 45 118

ATFTXT buffer 72

ATTACH macro instruction  
 compilation parameters 46  
 invoking the compiler 31  
 phase 02 processing 47

base and displacement element  
 phase 64 action 184  
 Procedure A-text format 435

base displacement data-name element  
 phase 64 action 186  
 procedure A1-text format 441

base locator number  
 (see also BL, BLL, SBL)

BL number  
 field in TGT 508  
 position in TGT 505

BLL number  
 field in TGT 509  
 position in TGT 505

Linkage Section (see BLL)

OPT option processing  
 optimizing register  
 assignments 168-170  
 phase 62 processing 168-170

SBL number  
 field in TGT 509  
 position in TGT 505

base locator reference  
 phase 6 processing 154  
 phase 64 processing 184  
 Procedure A-text format 435

basic lister format, IPTEXT 400

BASIS statement, phase 4 processing 51

BASISRTN routine 210

batch compilation (see BATCH option)

BATCH option  
 function 26  
 phase 02 processing for 48  
 phase 1B processing 66  
 phase 10 processing 56  
 phase 6 processing 141-142  
 phase 62 output 161  
 phase 65 processing for 189  
 PHZSW2 bit (COMMON) 334  
 resetting COMMON cells 324

BATCHSW cell (COMMON)  
 description 336  
 phase 02 processing 48  
 phase 1B processing 66  
 phase 10 processing 56

BCDCTR cell  
 description 332  
 use in compiler options 149

BCDISP cell (COMMON) 331  
BCDPN table  
    (see also TEST option)  
    description 466  
    phase 65 processing of 187-188  
BDAM access method  
    address elements 80  
    block address elements 80  
    buffer areas 82-83  
    constant definition elements 80  
BEGIN routine 71  
BEGPASS routine  
    description 88  
    flowchart 249  
between phase processing 31-33  
BGALLPN cell (COMMON) 338  
BGALLPRI cell (COMMON) 338  
BISAM access method  
    address elements 80  
    block address elements 80  
    buffer areas 80-83  
    constant definition elements 80  
BL  
    counter in COMMON 143  
    definition 542  
    field in TGT 509  
    phase 6 processing 154  
    phase 64 processing 184  
    position in TGT 505  
BL field (TGT), phase 62 counter for 163  
BL number (see BL)  
BL reference element 184  
BLASGTBL table  
    format 341-342  
    use in phase 62 169  
BLCHNG elements, phase 51 processing 136  
BLCTR cell (COMMON)  
    description 328  
    use in phase 22 76  
    use in phase 6  
        Data A-text processing 159  
        Procedure A-text processing 154  
        space allocation in TGT 143  
    use in phase 62 163  
    use in phase 64 184  
        space allocation 184  
        text processing 181  
BLDOBODO routine  
    diagram of 87  
    flowchart 261  
BLDRD routine 86  
BLL, definition  
    field in TGT 509  
    phase 6 processing 154  
    phase 64 processing 184  
BLL field (TGT), phase 62 counter for 163  
BLL number (see BLL)  
BLLCTR cell (COMMON)  
    description 326  
    use in phase 22 76  
    use in phase 6  
        Data A-text processing 154  
        space allocation in TGT 163  
    use in phase 62 163  
    use in phase 64 184  
block address elements  
    Data A-text format 409  
    phase 6 processing 154  
        phase 21 processing 80  
        phase 64 action 181  
block number element, procedure A1-text  
    format 441-442  
BLs, optimization of 169-170  
BLSRCH routine 169  
BLUSTBL table  
    format 342  
    use in phase 50 132  
BLVNTBL table  
    format 342  
    function 173  
    phase 62 processing 165  
BMBSRN routine 72  
branch instructions  
    phase 62 processing for 174  
    phase 63 processing for 175  
BRANCH routine  
    description 175  
    diagram of 583  
    flowchart 288  
BSAM access method  
    address elements 80  
    block address elements 80  
    buffer areas 80-83  
    constant definition elements 80  
BUF option  
    description 25  
    phase 02 47-48  
buffer pointer table 34  
buffer size  
    compilation 47,48  
    determination for object module 81  
buffers  
    compilation 47,48  
    diagnostic aids 487-488  
    object program 82,83  
BUFSIZE cell (COMMON)  
    description 336  
    use in phase 02 48  
BUGBLLNO cell (COMMON) 338  
BUGSTCRD cell (COMMON) 331  
BUGVLCNO cell (COMMON) 338  
building the PNLABTBL and GNLABTBL  
    tables 171  
BUSAGE routine 72  
CALL macro instruction  
    invoking the compiler 31  
    parameters passed to compiler 46  
    phase 01 processing 46  
    phase 02 processing 44  
CALL string 105  
calling sequence dictionary pointer element  
    phase 64 action 185  
    Procedure A-text format 436  
calling sequence displacement element  
    phase 64 action 185  
    Procedure A-text format 436  
card number  
    current  
        abnormal termination 487  
        phase 10 processing 55  
Debug-text  
    description 444  
    phase 6 processing 151-152

Procedure A-text  
   description 434  
   phase 6 processing 152  
   P0 format 412  
   P1 format 422  
   P2 format 429  
 card number elements  
   phase 63 processing of 177  
   phase 64 action 183  
 CARDINDX table  
   building of 188  
   format  
     compiler 343  
     debug data set 465  
 CARDLOC elements  
   debug text processing for 177  
   phase 65 processing 187  
 CARDNOTE save area, use in phase 65 189  
 CBL card  
   phase 02 processing 48  
   phase 1B processing 66  
   phase 10 processing 56  
 CD (see communication description)  
 CD dictionary entries, format 450,451  
 CD entries, data IC-text format 402  
 CD for initial input field (TGT)  
   description 508  
   location 505  
 CD-name reference, procedure IC-text  
   P1 format 419  
   P2 format 425  
 CD-names, phase 3 processing 95  
 CD-text, definition 542  
 CDECK option 29  
 CDLCCTR cell (COMMON) 331  
 CDSCNA routine 60  
 CDSECT routine flowchart 240  
 CDTEXT routine description 76  
 charts 197-308  
 checkpoint, debug processing 135  
 CHECKPT CTR field (TGT)  
   description 510  
   phase 62 counter for 164  
   position 505  
   use of CKPCTR counter (COMMON) 146  
 CHF-ROUT subroutine (Report Writer  
   subprogram)  
   definition 521  
   function 521  
   GENERATE statement coding 526  
 CHKTBL table (see CKPTBL table)  
 CKPCTR cell (COMMON)  
   description 326  
   use in phase 21 79  
   use in phase 6 146  
   use in phase 62 164  
 CKPTBL table  
   description 58-59  
   format 343  
   use in phase 21 79  
 clause compatibility 84  
 CLIST option  
   bit in COMMON 333  
   description 27  
   phase 02 processing 47  
   phase 62 output 161  
   phase 64 processing for 179  
   register assignment 483  
 CLOSE verb, debug processing 135  
 CLOSER routine, linkage code for 33  
 CLOSET routine 33  
 closing data sets 34  
 CMS interface routine  
   diagnostic aids 539-541  
   directories 535  
   environment 529  
   error messages 540  
   flowchart 536-538  
   functions 529  
   initialization 533  
   method of operation 532  
   operational considerations 531  
   option list 531  
   physical characteristics 529  
   program organization 535  
   relations with compiler 530  
   register usage 540  
   service routines 541  
 CNOP routine 583  
 CNTLINE cell (COMMON)  
   description 332  
   use in phase 00 48  
   use in phase 02 47  
 CNTLTBL table  
   description 190,191  
   format 344  
 COBEND routine, function 539  
 COBHAND routine  
   flowchart 537  
   function 539  
 COBOL command  
   (see also COBOL Prompter)  
   description of options 25-29  
   specifying compiler options 25  
 COBOL ID field (TGT)  
   description 508  
   location 505  
 COBOL INDICATOR (TGT)  
   description 507  
   location 505  
 COBOL Interactive Debug Program (see TEST  
   option)  
 COBOL library subroutines, definition 542  
 COBOL Prompter  
   invoking the compiler 19  
   NUM option 26  
 COBOL space, definition 542  
 COBOL subroutines, definition 542  
 COBOL verbs  
   code list 413-415  
   internal code list 413-415  
   phase 50 processing 120  
   phase 51 processing 135-137  
 COBOL word check 116  
 COBOL words, Procedure IC-text  
   internal code 416-418  
   P0 format 412  
   P2 format 428  
 CODE clause, RDSCAN routine processing 62  
 codes, error, use in phase 03 50  
 COLHIVAL cell (COMMON) 331  
 COLLITNO cell (COMMON) 331  
 COLLOVAL cell (COMMON) 331  
 COLUMN clause 62

COMFLOW cell (COMMON) 330  
 COMMAD cell (COMMON)  
 description 328  
 use in phase 10 57  
 COMMON

cells, description

ACCESSW 325  
 ADATAB 330  
 ADDR CARD 333  
 ADSTAT 325  
 AGETALL 328  
 AHEADER 338  
 AINSRT 325  
 ALIBEOD 339  
 ALIBSYNA 339  
 ALSTAM 325  
 ANAINF 325  
 AMICTR 330  
 AMILOC 331  
 AMOVDC 328  
 APLSCALL 331  
 APRIME 325  
 BATCHSW 336  
 BCDCTR 332  
 BCDISP 331  
 BGALLPN 338  
 BGALLPRI 338  
 BLCTR 328  
 BLLCTR 326  
 BUFSIZE 336  
 BUGBLLNO 338  
 BUGSTCRD 331  
 BUGVLCNO 338  
 CDLCCTR 331  
 CKPCTR 326  
 CNTLINE 332  
 COLHIVAL 331  
 COLLOVAL 331  
 COLLITNO 331  
 COMFLOW 330  
 COMMAD 328  
 COMPILES 332  
 CORESIZE 330  
 COS 325  
 CRDNUM 331  
 CURCRD 336  
 CURSGN 328  
 DATA BDSP 328  
 DATATBNM 330  
 DATE 331  
 DBGLOC 330  
 DBGODISP 332  
 DCBCTR 333  
 DCBNOXX 336  
 DCPTR 327  
 DECBCT 336  
 DEFCNT 333  
 DICADR 327  
 DICND1 327  
 DICND2 326  
 DICND3 338  
 DICPTR 327  
 DICTNAME 331  
 DLSVAL 327  
 ERF4SW 327  
 ERRNUM 332  
 ERRSEV 327  
 ESDID 338

FIL5BUF 330  
 FIPLVL 338  
 FLOWSZ 328  
 GNCTR 325  
 GTLNG 329  
 IDBYTES 339  
 IDENTL 328  
 IDPH00-IDPH80 339  
 INDEX 331  
 INDEX1 328  
 INITSIZE 338  
 INTVIRT 331  
 IOPTRCTR 328  
 KALOUT 337  
 KKADS5 337  
 KKPGR70 338  
 KKPHOSW 337  
 KTRMNATE 337  
 LABELS 325  
 LCSECT 327  
 LIBBUF 339  
 LINECNTX 332  
 LINKCNT 337  
 LISTERSW 331  
 LOCCTR 325  
 LOCTMCTT 331  
 LNGBL 331  
 LNGDSP 331  
 LTLCTR 326  
 MAXBGITM 338  
 NODECTR 330  
 NUMINCR 333  
 OBODOTBN 330  
 ODOCTR 326  
 ONCTR 329  
 OPTINSW 333  
 OPTINSW1 333  
 OPTINSW2 333  
 OPTINSW3 333  
 OPTINSW4 333  
 OPTLSTR 339  
 OPTLVL 339  
 OPTSWV2 339  
 OUTLRECL 339  
 PARMAX 330  
 PFMCTR 329  
 PHZERR 338  
 PHZSW 333  
 PHZSW1 333  
 PHZSW2 334  
 PHZSW3 334  
 PHZSW4 334  
 PHOSW 339  
 PH1BYTE 335  
 PH25SW 338  
 PH6ERR 329  
 PNCTR 325  
 PRBLDISP 325  
 PRBLNUM 330  
 PRINTBUF 338  
 PROCCTR 330  
 PROGID 325  
 PROGSW 338  
 PSVCTR 329  
 PTYNO 327  
 RELADD 325  
 RELLOC 329  
 RELSPACA 337

COMMON cells, description (continued)

RGNCTR 327  
 RPNCTR 328  
 RPTSAV 327  
 SA2CTR 327  
 SA3CTR 332  
 SBLCTR 326  
 SDSIZ 328  
 SEGLMT 328  
 SEQERR 326  
 SPACEX 336  
 SPACING 330  
 STAESW 336  
 SUBCTR 329  
 SWITCH 334  
 SWITCH1 337  
 SWITCH2 336  
 SWITV2 330  
 SYMSK 333  
 SYMSK1 334  
 SYMSK2 334  
 SYMSK3 334  
 SYNADR01 332  
 SYSTDD 338  
 SYSTX 328  
 TAMNAD 325  
 TIB (Table Information Block) 325  
 TMCNTBSZ 331  
 TSMAX 326  
 TS2MAX 326  
 TS3MAX 328  
 TS4MAX 328  
 VCONDISP 332  
 VIOVIRN 331  
 VIRCTR 325  
 VLCCTR 326  
 VNCTR 329  
 VNILOC 329  
 VNLOC 329  
 VTINITVN 331  
 V2BUGSW 338  
 WCMAX 326  
 WSDEF 327  
 XSACTR 329  
 XSWCTR 329  
 counters used for TGT 143,144  
 definition 542  
 function  
   overall design of compiler 21  
   phase 00 44  
 Program Global Table (PGT), relationship  
 to 324  
 register usage 324  
 Task Global Table (TGT), relationship  
 general 324  
 phase 6 142-146  
 use in  
   phase 00 44  
   phase 02 48  
   phase 1B 66  
   phase 10 56  
   phase 21 79,80  
   phase 3 88  
   phase 4 104  
   phase 50 127  
   phase 51 133-137  
   phase 6 148-152

Communication Description (CD)  
   phase 10 processing for 60  
   phase 20 processing 73  
 communication section dictionary entries,  
   phase 22 processing 77  
 communications area (see COMMON)  
 compilation directives  
   DEBUG card 105  
 compilation parameters (see options,  
   compilation directives)  
 compile-time  
   arithmetic 125-126  
   STATE bit (TGT) 506  
 COMPILED POINTER field (TGT)  
   description 508  
   location 505  
 compiler  
   COMMON, use of 21  
     (see also COMMON)  
   control information 25-29  
     (see also options; compilation  
     directives)  
   data sets 21  
   design 20-21  
   dictionary, use of 21  
     (see also dictionary)  
   directives (see compilation directives)  
   error handling (see error handling in  
     compilation)  
   generated procedure-name (see GN)  
   initialization 47  
   input 20  
   options 25-29  
     (see also options)  
   output 20  
   overview 549  
   parameters (see options; compilation  
     directives)  
   phases 21-25  
     (see also phase 00; phase 01; phase  
     02; phase 03; phase 05; phase 06;  
     phase 08; phase 1B; phase 10; phase  
     12; phase 20; phase 22; phase 21;  
     phase 25; phase 3; phase 4; phase  
     45; phase 50; phase 51; phase 6;  
     phase 62; phase 63; phase 64; phase  
     65; phase 6A; phases 70, 71, and 72,  
     phase 80)  
   physical structure 549  
   relationship to operating system 19  
   storage requirements 29-30  
   structure 549  
   tables 21  
     (see also tables used by compiler)  
 texts 399-444  
   (see also Data A-text; Data IC-text;  
   Debug-text; dictionary entries;  
   E-text; Listing A-text; Optimization  
   A-text; Procedure A-text; Procedure  
   A1-text; Procedure IC-text: P0  
   format, P1 format, P2 format; XREF  
   text)  
 COMPILES cell (COMMON) 332  
 completing dictionary entries  
   description 76-77  
   phase 21 processing 79,80  
   phase 22 processing 74  
 COMPUT routine 111



COMPUTE statement 111-113  
 COMSCT routine  
     flowchart 235  
     function 73  
 CONDIS table  
     description 165  
     format 344  
     literal allocation 150-151  
     optimizing DISPLAY literals 147  
     segmented program processing 155  
     use in literal allocation 168  
 condition-name dictionary entries,  
     format 454  
 condition-names 95  
 conditions causing ABENDs 483-484  
 Configuration Section 57  
 constant A-text 127  
 constant definition elements  
     Data A-text format 414  
     description 80,81  
     phase 21 processing 80,81  
     phase 64 action 182  
 constructing procedure A1-text 175  
 CONTBL table  
     format 344  
     literal allocation 151  
     optimizing literals 148  
     phase 62 processing 165  
     segmented program processing 155  
     use in literal allocation 168  
 control breaks in segmentation 134  
 control card for linkage editor  
     NAME option 26-27  
     phase 60 processing 141  
 CONTROL clause, RDSCAN routine  
     processing 62  
 control information (see options,  
     compilation directives)  
 CONTROL record  
     definition 542  
     description 190-191  
 control-field save-area names 61  
 Conversational Monitor System (see CMS  
     interface)  
 COPY statement, phase 4 processing 51  
 COPYPROC routine 212  
 COPYRN routine 51  
 COPYRTN routine 211  
 CORESIZE cell (COMMON)  
     description 330  
     use in phase 00 48  
     use in phase 02 48  
 CORRESPONDING options 89-91  
 CORRTRN routine  
     CORRESPONDING option 89,88  
     phase 3 operations 89-91  
 COS cell (COMMON) 325  
 COUNT CHAIN ADDRESS field (TGT)  
     description 508  
     location 505

COUNT LINKAGE AREA field (PGT)  
     description 510  
 COUNT location 510  
     option 29  
 COUNT TABLE ADDRESS field (TGT)  
     description 508  
     location 505  
 counters  
     AMICTR 330  
     BLCTR  
         Data A-text processing 159  
         description 328  
         Procedure A-text processing 154  
         TGT space allocation 143-144  
     BLLCTR  
         Data A-text processing 154  
         description 326  
         space allocation in TGT 143-144  
     CKPCTR  
         description 326  
         use in phase 21 79  
         use in phase 6 146  
     COMPILES 332  
     DCBCTR  
         address and constant definition  
         elements 80  
         Data A-text processing 159  
         DCBADR allocation 151  
         description 333  
         FD dictionary entries 79  
     DECBCT  
         address and constant definition  
         elements 80,81  
         Data A-text processing 159  
         description 336  
         FD dictionary entries 79  
         TGT space allocation 163  
     ERRNUM  
         description 332  
         phase 6 output 142  
     GNCTR  
         DEBUG CARD processing 106  
         Declaratives processing 70  
         description 325  
         GN allocation 150  
 INDEX1  
     description 328  
     TGT space allocation 144  
 LOCCTR  
     block and working-storage section  
     address elements 80  
     description 325  
     phase 6 output 142  
     Procedure A-text processing 151  
     segmented programs 153

counters (continued)

LTLCTR  
 description 326  
 phase 50 processing 132  
 NODECTR 330  
 ODOCTR 326  
 ONCTL  
 description 509  
 phase 51 processing 137  
 ONCTR  
 description 329  
 phase 51 processing 137  
 phase 6 processing 143  
 PARMAX  
 description 330  
 phase 6 processing 143  
 PFMCTL 509  
 PFMCTR  
 description 329  
 phase 4 processing 109  
 phase 6 processing 143  
 PNCTR  
 description 329  
 phase 1B processing 66  
 phase 51 processing 135  
 phase 6 processing 150  
 PROCCTR 330  
 PSVCTR  
 description 329  
 phase 6 processing 143  
 RGNCTR 327  
 RPTSAV  
 description 327  
 phase 6 processing 146  
 SA2CTR  
 description 327  
 phase 6 processing 164  
 SA3CTR  
 description 332  
 phase 6 processing 146  
 SBLCTR  
 description 326  
 phase 6 processing 143  
 SBSCTR 143  
 SEQERR 326  
 SUBCTR  
 description 329  
 phase 51 processing 137  
 TSMAX  
 description 326  
 phase 50 processing 163  
 phase 6 processing 163  
 TS2MAX  
 description 326  
 phase 6 processing 143  
 TS3MAX  
 description 328  
 phase 6 processing 143  
 TS4MAX  
 description 328  
 phase 6 processing 143  
 VIRCTR  
 description 325  
 phase 50 processing 131  
 phase 6 processing 148

VLCCTR  
 description 326  
 phase 6 processing 143  
 VNCTR  
 description 329  
 phase 6 processing 150  
 VNLOC  
 description 329  
 Procedure A-text processing 154  
 TGT space allocation 143  
 XSACTR  
 description 329  
 phase 6 processing 143  
 XSWCTR  
 description 329  
 phase 6 processing 143  
 phase 51 processing 137  
 CRDNUM cell (COMMON) 331  
 critical program breaks  
 Data IC-text format 404  
 definition 542  
 Procedure IC-text  
 P0 format 412  
 P1 format 421  
 P2 format 428  
 cross-reference listing  
 alphabetically ordered 191  
 compiler options 25-29  
 phase 6A processing 190-191  
 source ordered 190  
 CSECT names in phases 317-320  
 CSYNTAX option  
 description 26  
 E-text processing 192  
 phase 00 processing for 45  
 phase 02 processing for 47  
 phase 21 processing 79  
 phase 3 processing 96  
 phase 4 processing 116  
 phase 50 processing 119  
 phase 51 processing 133  
 SYSUT4 contents with 158  
 CTB-ROUT routine  
 description 520  
 GENERATE statement processing  
 first statement 525  
 subsequent statements 526  
 generation of 64  
 use of CTL.LVL counter 522  
 CTF-ROUT routine  
 CTL.LVL counter 522  
 description 521  
 FRS.GRP switch 523  
 GENERATE statement processing 526  
 generation of 64  
 locating routine in object module 528  
 logic of Report Writer  
 subprogram 516,517  
 CTH-ROUT routine  
 description 521  
 FRS.GRP switch 524  
 GENERATE statement processing  
 first statement 525  
 subsequent statement 526  
 generation of 64

locating routine in object module 528  
 logic of Report Writer  
     subprogram 516,517  
 CTL.LVL (Report Writer data-name) 522  
 CTLTBL table 345  
 CURCRD cell (COMMON)  
     description 336  
     use in phase 10 56  
     use in phase 4 104  
 CURGCN cell 56  
 current card number  
     abnormal termination 487  
     phase 1A processing 56  
 CURSGN cell (COMMON) 328  
 CVIRTB table  
     format 345  
     optimizing storage for PGT 148,149  
     Segmented program processing 155  
     use for compiler options 149  
     use in phase 62 166  
 CO routine diagram of 583  
 C1REF routine flowchart 292

Data A-text  
     definition 542  
     formats 409-410  
     generation of 77  
     input/output operations 37-43  
     phase 20 processing 72  
     phase 21 processing 83  
     phase 6 processing  
         description 22  
         flowchart 280  
     phase 64 action for 181  
     phase 64 processing 179  
         description 179  
         flowchart 294  
 DATA AREA field (object module) 502  
 DATABDSP cell (COMMON) 328  
 Data Control Block (see DCB)  
 Data Division  
     flow 221  
     general description of processing 20  
     phase 10 processing 59-60  
     glossary 88  
 Data Event Control Block (see DECB)  
 data management for compiler (see  
     input/output requests)  
 data operand, definition 543  
 DATA record  
     definition 543  
     description 190,191  
 Data IC-text  
     definition 542  
     formats 402-404  
     input/output operations 37-43  
 LD-text  
     definition 544  
     description 59  
     phase 10 processing 59  
     phase 21 processing 79  
     phase 21 processing 84  
 data set activity  
     CMS interface routine 439  
     compiler 37-43

data-name DEF-text element  
     A-text generation 79  
     phase 64 action 182  
 data-name definition elements, DEF-text  
     format 443  
 data-name information for UNSTRING elements  
     created by phase 45 118  
     procedure IC-text 426  
 data-name references, Procedure IC-text  
     P1 format 420  
     P2 format 426  
 data-name subscripts  
     phase 50 processing 121,122  
     Procedure IC-text format 426  
 data-names for Report Writer (see Report  
     Writer)  
 DATATAB table  
     built by phase 25 86  
     format 458-464  
 DATATBL table  
     description 190,191  
     format 346  
 DATATBNM cell (COMMON) 330  
 DATE cell (COMMON)  
     description 331  
     use in phase 02 47  
     use in phase 10 57  
 DATE-COMPILED clause 57  
 DBG R11SAVE field (TGT)  
     description 508  
     location 505  
 DBGFLPT bit (TGT) 506  
 DBGLOC cell (COMMON) 330  
 DBGTL table  
     format 346-347  
     phase 4 processing 105  
 DBGTEST routine  
     description 135  
     flowchart 272  
 DBGTXT table  
     description 100  
     format 347  
 DBGDISP cell (COMMON) 332  
 DC definition elements  
     format 435  
     phase 64 action 184  
 DCB (Data Control Block)  
     address elements  
         creation 80,81  
         Data A-text format 409  
         description 80,81  
     building for object module  
         address and constant definition  
         elements 80,81  
         FD dictionary entries 79  
         manipulation for compiler files 44  
 DCB address element, phase 64 action 181  
 DCBADR field (PGT)  
     description 511  
     location 510  
     phase 62 allocation 169  
     phase 64 processing for 181  
 DCBCTR cell (COMMON)  
     address and constant definition  
     elements 80,81  
     Data A-text processing 159  
 DCBADR allocation 150  
 description 333

DCBCTR cell (COMMON) (continued)  
 FD dictionary processing 79  
 use in phase 62 168

DCBNOXX cell (COMMON) 336

DCPTR cell (COMMON) 327

DDBG R14SAVE field (TGT)  
 description 507  
 location 505

DDSCN routine  
 communication section processing 60  
 Data Division processing 59  
 flowchart 225  
 phase 10 overview 56

DEBUG BLL field (TGT)  
 description 508  
 location 505

DEBUG card  
 description (TGT) 508  
 location (TGT) 505  
 use in phase 4 105

debug data set  
 description 455  
 format 455-469  
 phase 25 processing for 85

DEBUG LINKAGE AREA field (PGT)  
 allocation for 166  
 description 510-511  
 location 510

DEBUG MAX field (TGT)  
 description 508  
 location 505

debug options  
 COBOL Interactive Debug Program (see  
 TEST option)  
 phase 65 processing 187-189  
 TGT allocation for 164

DEBUG PTR field (TGT)  
 description 508  
 location 505

DEBUG TABLE field (TGT)  
 description 510  
 location 505  
 phase 65 processing 189

DEBUG TABLE PTR field (TGT)  
 description 507  
 location 505  
 phase 65 processing 189

debug-text  
 construction in phase 63 176  
 data sets used for 151  
 definition 543  
 description 151  
 format 444  
 input/output operations 37-43  
 phase 6 processing 151  
 phase 63 processing of 176-177  
 phase 65 processing 187-189

DEBUG TRANSFER field (TGT)  
 description 508  
 location 505

DEBUGGING field (TGT)  
 description 508  
 location 505

debugging (see diagnostic aids)

DEBUG VLC field (TGT)  
 description 508  
 location 505

DECB (Data Event Control Block)  
 address elements  
 Data A-text formats 409  
 creation 80,81  
 description 80,81  
 building for object module  
 address and constant definition  
 elements 80,81  
 FD dictionary entries 79  
 manipulation for compiler files 44  
 (see also DECBCT cell)

DECB address element, phase 64 action 181

DECBADR field (TGT)

DECBCT cell (COMMON) 336

DECBCT counter (phase 6) 143  
 description 508  
 location 505  
 phase 62 counter for 164

DECBCT cell (COMMON)  
 Data A-text processing 159  
 description 336  
 TGT space allocation 143  
 use in phase 2 79  
 use in phase 62 164  
 use in phase 64 181

DECIMAL-POINT IS COMMA clause  
 COMMAD cell (COMMON) 328  
 phase 10 processing 57

DECK option  
 bit in COMMON 333  
 description 28  
 phase 02 processing 47  
 phase 62 output 161

Declarative Section  
 description 70  
 error declaratives 70  
 label declaratives 70

DEF-text  
 compiler processing 24-25  
 definition 543  
 formats 444  
 input/output operations 37-43  
 phase 22 processing 75  
 phase 6A processing 190,191  
 phase 64 action for 181  
 phase 64 processing of 179  
 phases involved 316

DEFCNT cell (COMMON) 333

DEFILD11 routine  
 description 171  
 diagram of 581

DEFSBS table 347

delimiter, definition 543

delimiter pointer  
 definition 543  
 format 447

design of compiler  
 diagram 581  
 general description 20

destination table entry, Data IC-text 404

DESTROY element  
 function 120  
 use in phase 62 169

DET-ROUT routine  
 description 522  
 FRS.GRP switch 523

GENERATE statement logic flow 526

logic of Report Writer  
   subprogram 527,316  
   phase 1B processing 61  
 DETBL table  
   description 66  
   format 347-348  
   output of phase 12 63  
 DEVTYPE macro instruction 48  
 diagnostic aids  
   ABEND codes 483-484  
   abnormal termination 31,32  
   buffers 487-488  
   CE worksheet 490-491  
   compiler error messages 485-486,483  
   current phase 486-487  
   description 470-491  
   registers  
     assignment 483  
     saving 487  
     usage by phases 471-482  
   system error recovery program 470  
   tables 313-315,488  
   terminal error conditions 45  
   version of compiler 486  
 DICADR cell (COMMON) 327  
 DICND1 cell (COMMON)  
   description 327  
   use in phase 3 88  
 DICND2 cell (COMMON) 326  
 DICND3 cell (COMMON) 338  
 DICOT table  
   format 348  
   input to phase 25 85  
   organization of the dictionary 492  
   use in phase 1B 67  
 DICPTR cell (COMMON) 327  
 DICSPC routine 499  
 DICTBD routine  
   description 72,74  
   flowchart 244  
 dictionary  
   attributes, definition 453  
   definition 453  
   description 492  
   entries (see dictionary entries)  
   handling routines (see ACCESS routines)  
   organization 492,493  
   pointer, definition 453  
   spill  
     compiler data set activity 37-43  
     switch in COMMON 335  
     TAMEIN routine 498  
     TBSPIII routine 500  
   storage for 493  
   dictionary entries  
     (see also dictionary)  
   attributes  
     descriptions 446-454  
     phase 1B processing 66-68  
   base locator (see base locator)  
   building 74  
   completing  
     phase 21 processing 79,80  
     phase 22 processing 76  
   count field 79  
   FD  
     (see also FD dictionary entries)  
     completing 77  
     phase 21 processing 79,80  
     preprocessing 76-77  
   formats 445-454  
   handling routines (see ACCESS routines)  
   LD 72  
     (see also LD dictionary entries)  
   major code  
     definition 544  
     FD dictionary entries 79  
   minor code, definition 544  
   partial 76  
   phase 02 processing 47  
   phase 1B processing 66-68  
   phase 21 processing 79,80  
   phase 22 74-77  
   phase 3 processing 95  
   RD 76  
     (see also RD dictionary entries)  
   REDEFINES clause 77  
   routines (see ACCESS routines)  
   SD  
     (see also SD dictionary entries)  
     completing 77  
     preprocessing 76  
     storage allocation 47  
   dictionary preprocessing 74-76  
   DICTNAME cell (COMMON) 331  
   direct A-text 127  
   direct indexing 123  
   DIRECTOR routine 74  
   directories, microfiche 317-323  
   discontinuity elements  
     phase 63 processing of 177  
     phase 65 processing 187  
   display literal definitions  
     optimization 147,165  
     Optimization A-text format 439  
   PGT field 512  
     phase 6 processing 147  
   DISPLAY LITERAL field (PGT)  
     description 512  
     location 510  
   DISPLAY verb translator routine 139  
   DLSVAL cell (COMMON) 327  
   DMAP option  
     bit in COMMON 333  
     description 27  
     phase 02 processing 47  
     phase 62 output 161  
     register assignments 483  
   DMSAUPD routine, function 535  
   DMSCBD module  
     external symbol directory 535  
     function 535  
     load module directory 535  
   DMSCOB routine 533  
     (see also CMS interface routine)  
   DMSERR routine, called by DMSCOB 541  
   DMSERS routine  
     called by DMSCOB 541  
     function 535  
   DMSFLD routine  
     called by DMSCOB 541  
     function 535  
   DMSFNPA routine  
     called by DMSCOB 541  
     function 535  
   DMSGND routine 533

DMSILB routine, ENTRY card punched for 533  
 DMSLADW routine 541  
 DMSSBS routine 541  
 DMSSMN routine  
     called by DMSCOB 541  
     function 535  
 DMSSTT routine 541  
 DNTOR1 routine 138  
 DOFINIS routine, function 539  
 DOP1 workarea, use in phase 45 118  
 DRPLTBL table  
     format 348-349  
     function 169-170  
 DRPTBL table  
     function 169-170  
     format 349  
 DSPLAC routine (phase 6) 144  
 DSPLAC routine (phase 62) 164  
 DTAB table  
     description 100  
     format 349  
 dump, producing a 29,470  
 DUMP option 29  
     ABEND conditions 483-484  
     description 29  
 DYNAM option  
     allocation of virtuals for 149  
     description 28  
     phase 02 processing 47  
     phase 51 processing 139  
     phase 6 processing for 148-149  
     phase 62 virtual EBCDIC names allocation  
         for 167  
     virtual allocation for 167  
  
 E-point name, phase 22 processing 62  
 E-text  
     definition 543  
     description 25  
     format 442  
     input/output operations 37-43  
     introduction 25  
     phase 20 71,72  
     phase 4 116  
     phase 51 133  
     phase 6  
         action taken 159  
         suppression of output listing 142  
         SYSUT4 processing 158  
     phase 64 action for 181  
     phase 64 processing of 179  
     phase 70 192-193  
     phases involved 316  
 E.nnnn (Report Writer data-name)  
     column clauses 523  
     nonstandard data-names 523  
 EACTBL table 193  
 EBCDIC card name element, phase 64 183  
 EBCDIC data-name reference element  
     phase 64 action 185  
     Procedure A-text format 435  
 EBCDIC name, Procedure IC-text format 411  
 EBCDIC procedure-name generator, Procedure  
     A-text format 434  
 EJECT routine 33  
 elementary item processing, phase 20 72  
  
 ELIODO routine 78  
 element (text), definition 543  
 ELSE clause 113  
 ENCODE routine 585  
 ENDJOB option  
     description 28-29  
     parameter for 47  
     switch in COMMON 334  
 ENDOFTBL routine  
     diagram 585  
     function 189  
 ENDPTX routine (phase 6), diagram of 581  
 ENDP1 routine flowchart 261  
 ENDP16 routine, flowchart 261  
 ENDSEG elements  
     phase 63 processing of 177  
     phase 65 processing 187  
 ENDUSE verb, debug processing 135  
 ENTDEL routine 494-495  
 ENTDRP routine (phase 62) 169  
 ENTDRPL routine 169  
 ENTNAM routine 494  
 ENTPTR routine 494  
 ENTPT01 routine 174  
 ENTRDATA routine  
     description 86  
     flowchart 250  
     entry (table), definition 543  
     entry points in phases 317-323  
 ENTRY-SAVE field (TGT)  
     description 507  
     location 505  
 Environment Division, phase 10  
     processing 57-58  
 ENVSCN routine  
     Environment Division processing 57  
     flowchart 224  
     phase 10 introduction 56  
 ENVTBL table  
     description 58  
     format 350-351  
 EOF routine (phase 63), diagram of 583  
 EOFON2 routine  
     description 188  
     diagram 585  
 EOFRTN routine 102  
 EOF2 routine, diagram 585  
 equate string  
     built by phase 51 134  
     GN 134  
     optimizing PNs and GNs 146  
     phase 6 processing 144  
     PN 134  
     use in Optimization 144  
 ERAS routine, function 539  
 ERF4SW cell (COMMON) 327  
 ERRNUM cell (COMMON)  
     description 332  
     passing E-text 141  
     phase 00 processing 142  
 error, definition 543  
 error codes, use in phase 03 50  
 error declaratives  
     description 70  
     debug processing 133  
 error handling in compilation  
     clause compatibility 84  
     compilation in parameter errors 49

compiler errors 483  
 input/output errors  
   diagnostic aids 470  
   phase 02 processing 49  
   SYNAD routine 34  
   terminal 45  
 message generation, flowchart 303  
   (see also E-text)  
 severity 31  
 source program errors  
   phase 10 processing 56  
   phase 20 processing 73  
   phase 22 processing 78  
   phase 4 processing 116  
   terminal 45  
   terminal error conditions 45  
 error messages  
   CMS interface routine 540  
   generation by phase 12 64  
   printed by phase 03 50  
 error message texts 192  
 ERROR routine  
   phase 3  
     description 96  
     operations diagram 597  
   phase 4 116  
 error symbols, Procedure IC-text  
   P0 format 412  
   P1 format 422  
 error text (see E-text)  
 ERRPRO routine 133  
 ERRSEV cell (COMMON) 327  
 ERRTBL table  
   format 352  
     E-text processing 159  
     general information 137  
     suppression of output listing 142  
   phase 7 processing 192  
   syntax-checking function 45  
   use in phase 3 95  
   use in phase 4 116  
   use in phase 50 119  
   use in phase 51 133  
   use in phase 6 142  
   use in phase 62 162  
   use in phase 64 181  
 ESD 149  
 ESD cards, definition 543  
 ESD-text, definition 543  
 ESDID cell (COMMON) 338  
 EVAL string 112-113  
 EVERY option 137  
 EXEC control card  
   compiler options 25  
   phase 00 processing 31  
 execution-time STATE bit (TGT) 506  
 EXHIBIT NAMED name, Procedure IC-text  
   P0 format 411  
   P1 format 421  
   P2 format 427  
 exit lists 80  
 external symbol dictionary  
   description 148  
   microfiche directory 321-323  
 FD dictionary entries  
   completing 77  
   format 447  
   phase 21 processing 79,80  
   preprocessing 76  
 FD entries  
   Data IC-text format 403  
   description 59  
 FD text  
   definition 543  
   description 79  
 FDECK option 29  
 FDEFCOB routine, function 539  
 FDTAB table  
   format 352  
   use in phase 21 79  
   use in phase 21 80  
   use in phase 22 75  
 FIB field (TGT)  
   description 508  
   location 505  
 figurative constant ALL references,  
   Procedure IC-text format 428  
 figurative constant references, Procedure  
   IC-text  
     P0 format 412  
     P1 format 422  
     P2 format 428  
 File Description dictionary entries (see FD  
   dictionary entries)  
 File Description entries 59  
 FILE-CONTROL paragraph 58  
 file information block field (TGT)  
   description 508  
   location 505  
 file-name DEF text element, phase 64  
   action 181  
 file-name reference elements  
   description 89  
   Procedure IC-text  
     P1 format 425  
     P2 format 429  
 file-name reference elements  
   phase 64 action 185  
   Procedure A-text format 436  
   Procedure IC-text format 419  
 File Section  
   description 59  
   dictionary entries  
     phase 22 77  
     phase 20 72  
 FILEDEF commands  
   description 532  
   issued for CMS interface 532  
 FILEST routine (phase 20) flowchart 234  
 FILEST routine (phase 21)  
   description 72  
   flowchart 246  
 FIL5BUF cell (COMMON) 330  
 FINDRW routine, function 539  
 FINDSSC routine 118  
 FIPLVL cell (COMMON) 338  
 FIPS  
   processing for phase 8s 194

FIPS (continued)  
 flowcharts 304-308  
 fixed Report Writer routines 515  
 FLAG option  
   bit in COMMON 333  
   phase 02 processing 47  
 floating-point literal references,  
   Procedure IC-text  
   P0 format 411  
   P1 format 421  
   P2 format 427  
 floating-point operations 124-125  
 FLOW option  
   bit in COMMON 333  
   description 26  
   phase 02 processing 47  
   phase 6 output 142  
   phase 62 output 162  
   phase 65 processing 187  
   Procedure A-text processing 152  
 flow trace option (see FLOW option)  
 flowcharts 197-308  
 FLOWSZ cell (COMMON)  
   description 328  
   use in phase 65 187  
 FLUSH routine  
   description 64  
   flowchart 230  
 FNTBL table  
   Data Division processing 59  
   format 352-353  
   input to phase 12 63  
   output of phase 12 63  
   phase 1B processing 69  
   phase 10 processing 59  
 forcing a dump 470  
 FORMLA routine 111,112  
 FOURTY8 routine, diagram of 583  
 fragment, program, definition 543  
 FREE element  
   function 120  
   use in phase 62 169  
 FREEMAIN macro instruction 498  
 FRS.GEN (Report Writer data-name),  
   description 522  
 FRS.GRP (Report Writer data-name),  
   description 523  
 FRTYPROC routine  
   description 187  
   diagram 585  
 FSECT routine flowchart 252  
 FSTXT routine  
   FD processing 79  
   phase 21 processing 80  
 FST000 routine 77,79  
 F2PROCS branch table, use in phase 65 187

GATXTV routine 127-128  
 GCNTBL table 353  
 GENERATE statement  
   FRS.GEN data-name 524  
   logic flow  
     first statement 525  
     subsequent statements 526  
   phase 1B processing of 61  
   response at execution time 523

special Report Writer verbs 524  
 1ST-ROUT routine 515  
 generated procedure-name (see GN)  
 generating data A-text 77  
 GENOP routine  
   condition-string processing  
     with VALUE clause 98  
     without VALUE clause 97  
   replacing names 95  
   phase 3 operations 88  
 GENOP routine, translation of P0 text 89  
 GET routine (phase 63)  
   control 175  
   diagram of 583  
 GETALL routine (TAMER) 500  
 GETBTBL table, use in phase 63 182  
 GETCRD routine  
   phase 1B 66  
   phase 10 57  
 GETDLM routine  
   description 56  
   phase 10 56  
   phase 12  
     call to GNSPRT routine 64  
     RDSCAN routine 62  
 GETMAIN macro instruction  
   dictionary storage 493  
   TAMER area 497,498  
 GETNXT routine (Phase 3) 89,90  
 GETNXT routine (phase 50)  
   flowchart 267,268  
   introduction 119  
 GETNXT routine (phase 51)  
   flowchart 273  
   introduction 133  
 GETPTR routine 494  
 GETWD routine 56,57  
 global table (see Task Global Table;  
   Program Global Table)  
 global table references, Procedure IC-text  
   format  
     type 1 430  
     type 2 430  
 global table standard area references 153  
 global table variable-located area  
   reference element  
     description 153  
     phase 64 action 184  
     Procedure A-text format 435  
 GLORET routine 88  
 GLOSRY routine  
   flowchart 252  
   glossary building 88  
   phase 3 operations 88  
 glossary (compiler)  
   processing for 88  
   symbols used in 142  
 glossary (for this book) 542-546  
 GN (compiler generated procedure-name)  
   allocation  
     phase 6 150  
     phase 62 167  
   definition  
     phase 64 action 184  
     Procedure A-text format 434  
   P0 text format 412  
   P1 text format 422  
   P2 text format 428



description, general 543  
 equate strings  
     building 134  
     Optimization A-text format 439  
     optimizing 146-147  
 error declaratives, Procedure IC-text  
     format 411  
 field (PGT) 511  
 generated procedure-name reference,  
     optimization A-text format 439  
 label declaratives, Procedure IC-text  
     format 411  
 number  
     phase 1B 70  
     phase 3 95  
     phase 4 105  
     phase 50 130  
     phase 6 150  
 optimizing 146-147  
     phase 1B processing 70  
     phase 4 processing 113  
     phase 50 processing 130  
     phase 51 processing 134  
     phase 6 processing 146  
 reference element  
     phase 64 action 184  
     Procedure A-text format 434  
     P0 text format 412  
     P1 text format 422  
     P2 text format 428  
 GN-VN element for PERFORM verb,  
     optimization A-text format 440  
 GNCALTBL table, format 354  
 GNCTR cell (COMMON)  
     description 325  
     use in phase 1B 70  
     use in phase 4 105  
     use in phase 6 149  
 GNDEF routine  
     diagram of 583  
     flowchart 289  
 GNDEFR routine 581  
 GNEQR routine 146  
 GNFWDBTB table  
     building of 171  
     format 354  
 GNLABTBL table  
     building of 171  
     format 354  
     use in phase 63 175  
     use of ACCUMCTR 171  
 GNLBDBTBL table 355  
 GNREF routine  
     diagram of 583  
     flowchart 292  
 GNSPRT routine  
     description 64  
     flowchart 231  
 GNTBL table  
     format 355  
     GN allocation 150  
     optimizing GNs 146-147  
     Procedure A-text processing 152  
 GNUREF elements  
     optimization A-text format 439  
     phase 50 processing 132  
 GNVNRTN routine 165  
 GOBACK statement, ENDJOB option for 28-29  
 GO string 106  
 GO TO DEPENDING ON call parameter element  
     phase 64 action 185  
     procedure A-text format 436  
 GO TO statement  
     phase 1B processing 69  
     with ALTER statements 105-108  
 GOTAVRBL routine  
     description 101  
     flowchart 257  
 GPLSTK table  
     format 355-356  
     use in phase 22 76  
 group item processing, phase 20 72  
 GRP.IND (Report Writer data-name) 522  
 GSPICT routine 72  
 GTEQ10K routine  
     description 188  
     diagram 585  
     flowchart 299  
 GTLNG cell (COMMON) 329  
 GVFNTBL table 357  
 GVNMTBL table 357  
 HASH table  
     dictionary organization 492,493  
     format 358  
     input to phase 25 85  
     phase 3 processing 90  
     TIB30 cell (COMMON) 325  
     usage 313-314  
     use in phase 1B 67  
 hierarchy of operators  
     definition 543  
     description 112  
 I-O-CONTROL paragraph 58  
 IC-text, definition 543  
     (see also Data IC-text; Procedure  
     IC-text)  
 IDBRK routine 104  
 IDBYTES cells (COMMON) 339  
 IDDBSCN routine  
     flowchart 223  
     Identification Division processing 57  
     phase 10 introduction 56  
 IDENT routine 104  
 identification, compiler version 486  
 Identification Division  
     compiler processing 20  
     phase 10  
         overview 56  
         processing 57  
 identifier constant 486  
 IDENTL cell (COMMON) 328  
 ID entry format, Data IC-text 104  
 idk field, phase 22 processing 76  
 IDLHN routine  
     ALTER statement processing 106  
     DEBUG Card processing 105  
     PERFORM statement processing 109  
     phase 4 overview 104  
 IDPH00-IDPH80 cells (COMMON) 339  
 IF statement 113

IF string  
   IF statement processing 113  
   PERFORM statement processing 109  
 IF verb analyzer routine  
   flowchart 260  
   IF statement processing 113  
   IF MESSAGE statement processing 113  
 IKFCBL00 (see phase 00; producing a storage dump)  
 IKFCBL01 (see phase 01)  
 IKFCBL02 (see phase 02)  
 IKFCBL03 (see phase 03)  
 IKFCBL04 (see phase 04)  
 IKFCBL05 (see phase 05)  
 IKFCBL06 (see phase 06)  
 IKFCBL08 (see phase 08)  
 IKFCBL1B (see phase 1B)  
 IKFCBL10 (see phase 10)  
 IKFCBL12 (see phase 12)  
 IKFCBL20 (see phase 20)  
 IKFCBL21 (see phase 21)  
 IKFCBL22 (see phase 22)  
 IKFCBL25 (see phase 25)  
 IKFCBL30 (see phase 3)  
 IKFCBL35 (see phase 35)  
 IKFCBL40 (see phase 4)  
 IKFCBL45 (see phase 45)  
 IKFCBL50 (see phase 50)  
 IKFCBL51 (see phase 51)  
 IKFCBL6 (see phase 6)  
 IKFCBL6A (see phase 6A)  
 IKFCBL62 (see phase 62)  
 IKFCBL63 (see phase 63)  
 IKFCBL64 (see phase 64)  
 IKFCBL65 (see phase 65)  
 IKFCBL70 (see phase 70)  
 IKFCBL71 (see phase 71)  
 IKFCBL72 (see phase 72)  
 IKFCBL80 (see phase 80)  
 incremented address elements  
   phase 63 processing of 176  
   phase 64 action 185  
   Procedure A-text format 435  
 incrementing the ACCUMCTR counter 171  
 INDEX cell (COMMON) 331  
 INDEX field (TGT)  
   description 509  
   location 505  
   phase 6 processing 143  
   phase 62 counter for 163  
   use in phase 50 123  
 INDEX table, phase 22 output 75  
 index-name format, Data IC-text 404  
 index-name references  
   description 123-124  
   dictionary entry format 454  
   Procedure IC-text  
     P0 format 421  
     P1 format 427  
 INDEXED BY clause 143  
 INDEX1 cell (COMMON)  
   description 328  
   use in phase 6 143  
   use in phase 62 163  
 indirect indexing 123,124  
 INDKEY table  
   format 358-359  
   phase 3 processing 88  
   SEARCH verb processing 92,93  
 INDXTB table 359-360  
 INDTBL table  
   description 58  
   format 358  
 IND2TBL table 359  
 initialization coding  
   description 157  
   flowchart 297  
   generation of  
     flowchart 281  
     phase 64 183  
 initialization of compiler 47  
 INITIATE statement 524  
 INITIATE verb, phase 1B processing for 61  
 INITSIZE cell (COMMON) 338  
 INIT1 routine (object module)  
   coding generation 157  
   description 501-502  
   location 501  
   written by phase 6 141,157  
 INIT2 routine (object module)  
   coding generation 157  
   description 513  
   location 501  
   written by  
     phase 6 141,157  
     phase 64 183  
 INIT3 routine (object module)  
   coding generation 157  
   description 513-514  
   location 501  
 RLDTBL table processing 157  
   written by  
     phase 6 141,157  
     phase 64 183  
 inline procedures, definition 544  
 input/output  
   (see also access methods)  
   buffer assignments 37-43  
   compiler  
     buffer contents 487-488  
     buffer processing 47,48  
     data set activity 37-43  
     error messages 483  
     errors from phase requests 34  
     linkage codes 33,34  
     phase requests 34  
     phase 00 operations 31  
     register usage 471-482  
     response to system error recovery 470  
     scanning routines 215  
     summary of phases 21-25  
     terminal errors 45  
   data set activity 37-43  
   errors  
     compiler messages 483  
     phase 02 processing 49  
     response to system recovery 470  
     result of phase requests 34  
     terminal 45  
   object module  
     (see also DCB; DECB)  
     buffer size 81  
     compiler processing for 80,81  
     data area 502  
     exit tests 502-504

requests  
   linkage codes for 33,34  
   phase 00 processing 31  
   phase 34  
   summary of phases 21-25  
 Input-Output Section 58,59  
 INSERT routine 500  
 INT-ROUT routine  
   COBOL word data-name processing 522  
   description 520  
   generation of 64  
   logic 516,517  
 Interactive Debug Program (see TEST option)  
 interlude routines  
   flow of control 35  
   linkage to data management 44  
   phase 00 processing 31,34  
 Intermediate A-text  
   data sets used for 119  
   definition 544  
   description 24  
   phase 50 processing 119  
   phase 51 processing 133  
 Intermediate E-text  
   data sets used for 119  
   description 119  
   definition 544  
   phase 50 processing 119  
   phase 51 processing 133  
 intermediate results  
   definition 544  
   work area for 124  
 intermediate result references, Procedure  
   IC-text format 429  
 internal compiler text 399-444  
   (see also ATF-text; ATM-text; Data  
   A-text; Data IC-text; Debug-text;  
   DEF-text; E-text; Optimization A-text;  
   Procedure A-text; Procedure A1-text;  
   Procedure IC-text; XREF-text)  
   ATF-text format 408  
   ATM-text format 433  
   Data A-text format 409  
   Data IC-text format 402  
   Debug-text format 444  
   description 399  
   E-text format 442  
   Optimization A-text format 439  
   Procedure A-text format 434  
   Procedure A1-text format 441  
   Procedure IC-text  
     P0 format 411  
     P1 format 419  
     P2 format 425  
   types produced by each phase 316  
   XREF text format 443  
 internal text formats 399-444  
 interphase routines (see interlude  
   routines)  
 INTxx routines (see interlude routines)  
 INTVIRT cell (COMMON) 331  
 IOPTRCTR cell (COMMON) 328  
 IPTTEXT  
   formats 400-401  
   generation of 53  
 IPTTEXT ITEM processors 217  
 issuing CMS FILEDEF commands 532  
  
 KALOUT cell (COMMON) 337  
 KEYTAB table 360  
 KEYTBL table  
   format 360-361  
   SEARCH processing 114  
 KILSUB routine  
   description 132  
   flowchart 270  
 KKADSS cell (COMMON) 337  
 KKPGR70 cell (COMMON) 338  
 KKPHOSW cell (COMMON) 337  
 KTRMNATE cell (COMMON)  
   description 337  
   use in phase 03 50  
  
 label declaratives  
   debug processing 135  
   description 70  
 LABEL RET field (TGT)  
   description 507  
   location 505  
 LABELS cell (COMMON) 325  
 LABTBL table  
   format 361  
   input to phase 20 71  
 LANGLVL option 29  
 language analysis routine 214  
 LATACP routine  
   description 496  
   dictionary organization 493  
 LATGRP routine 496  
 LATRNM routine 495  
 LATRPT routine 495  
 LCOL1 option 29  
 LCOL2 option 29  
 LCSECT cell (COMMON) 327  
 LD dictionary entries  
   description 72  
   format 450  
   preprocessing 71  
 LD entries, Data IC-text format 402  
 LD-text  
   definition 544  
   description 59  
 LDELNM routine 495-496  
 LDTEXT routine  
   description 73  
   flowchart 236  
   phase 20 control 72  
 LDTXT routine, flowchart 242  
 LENGTH OF VN TBL field (TGT)  
   description 507  
   location 505  
 LIBBUF cell (COMMON) 339  
 LIB option  
   data set usage for 37-43  
   description 25-26  
   flowchart 207  
   parameters 47  
   phase 10 processing 57  
   switch for 295  
 LIN.NUM data-name 523  
 LIN.SAV data-name 523  
 LINE clause, input to PROC01 routine 56  
 LINE-COUNTER data-name 522

LINECNT option  
     description 28  
     phase 02 parameters 47  
 LINECNTX cell (COMMON) 332  
 LINKCNT cell (COMMON) 337  
 LINK macro instruction  
     interphase processing 31  
     parameters 46  
     passing control to compiler 31  
     phase 02 47  
 LINKA routine 31  
 linkage editor  
     introduction 19  
     phase 6 overview 22  
     processing of CMS compiled program 533  
 Linkage Section  
     phase 10 processing 60  
     phase 22 processing 77  
 LINKB routine 35  
 LINKCNT cell (COMMON)  
     current phase 487  
     description 337  
     interphase processing 34  
     returning control to system 31  
     use in phase 4 116  
 LINKNAME cell 487  
 LINKPH1 routine 71  
 LINKST routine (phase 20), flowchart 235  
 LISTERSW cell (COMMON) 331  
 LISTING filename 532  
 listing  
     suppression of 142  
     symbols used in 142  
 Listing A-text  
     phase 60 processing 155  
 literal allocation, phase 62 168  
 literal definitions  
     A-text generation 127,131  
     optimization 147  
     optimization A-text format 439  
     phase 51 processing 139  
 LITERAL field (PGT)  
     description 511-512  
     location 510  
 literal reference element  
     phase 64 action 184  
     Procedure A-text format 435  
 literal subscripts 121,122  
 literals  
     description 511-512  
     optimization of 165  
 load module 46  
 load module microfiche directory 317-323  
 LOAD option  
     description 26  
     phase 02 parameters 47  
     phase 62 output 161  
     switch for 333  
 LOCCTR cell (COMMON)  
     description 325  
     use in ADINCR routine 176  
     use in phase 21 80  
     use in phase 22 76  
     use in phase 6 142-144  
     use in phase 63 175-178  
     use in RPT-ORIGIN processing 177  
 LOCNXT routine  
     description 495  
     phase 25 processing 86  
 LOCTMCTT cell (COMMON) 331  
 LONGTGT bit (TGT) 506  
 LNGBL cell (COMMON) 331  
 LNGDSP cell (COMMON) 331  
 LSECT routine flowchart 239  
 LST-ROUT routine  
     generation of 64  
     GN number 528  
     phase 1B processing 61  
 LSTCOMP option 29  
 LSTONLY option 29  
 LTLCTR cell (COMMON)  
     description 326  
     use in phase 50 132  
     use in phase 6 150  
     use in phase 62 168  
 LTLDIS routine  
     literal optimization 169  
     phase 6 169  
     phase 62 165  
 LTLRTN routine  
     literal optimization 169  
     phase 6 169  
     phase 62 165  
 LTLTBL table  
     format 361  
     LITERAL allocation 150-151  
     literal optimization 169  
     phase 62 processing 165,168  
     processing Procedure A-text  
         elements 153  
     use in phase 62 168  
 LVL option 28  
 L120 option 29  
 L132 option 29  
 macro instructions  
     ATTACH 31,46  
     CALL 31,46-47  
     FREEMAIN 498,499  
     RETURN 32  
     XCTL 31,46-47  
 MACRO routine, diagram of 583  
 macro-type instruction element  
     optimization A-text format 440  
     phase 64 action 183  
     Procedure A-text format 434  
 Main Free Area, definition 544  
 major code, definition 544  
 master of an OCCURS clause with the  
     DEPENDING ON option, definition 544  
 MAPLOC routine  
     PGT storage allocation 148  
     TGT storage allocation 144  
 MASTAM table 497-498  
 MASTODO table  
     format 361-362  
     input to phase 25 85  
     output of phase 22 75  
 MAXBGITM cell (COMMON) 338  
 MESSAGE condition, phase 50 processing  
     for 132  
 message definitions, E-text format 442  
 message parameters, E-text format 443

microfiche directories  
 external symbol dictionary 321-323  
 load module 317-323  
 minor code  
 definition 544  
 description 450  
 mode of operation for arithmetic strings 126  
 MOVDIC routine 499  
 MOVE statement 104  
 MOVE string 104  
 MOVE verb analyzer routine  
 subscript references 123  
 verb string processing 120

N.nnnn data-name 504  
 NAME option  
 bit in COMMON 334  
 description 26  
 phase 02 parameters 47  
 phase 6 output 137  
 phase 62 output 161  
 phase 65 processing for 189  
 nested IF statements 113  
 NEWBLOCK routine 581  
 NEXT GROUP clause 64  
 NOBLST routine 171  
 NODECTR cell (COMMON) 330  
 nondata operand, definition 544  
 NOTE routine 44  
 NPPTBL table 362  
 NUM option  
 bit in COMMON 334  
 description 26  
 GETWD routine 56  
 phase 02 parameters 47  
 numeric literal references, Procedure IC-text  
 P0 format 411  
 P1 format 420  
 P2 format 427  
 NUMINCR cell (COMMON) 333  
 NXTOPTN routine, function 539

object deck 28  
 object hierarchy, definition 544  
 object module  
 definition 544  
 description 501-514  
 fields  
 COUNT table 512  
 DATA AREA 502  
 EXIT lists 501-504  
 INIT1 501-502  
 INIT2 513  
 INIT3 513  
 locations 501  
 PGT 510  
 (see also Program Global Table)  
 PROCEDURE 512  
 PROCTAB table 514  
 Q-Routines 512

RPT 512  
 SEGINDX table 514  
 TGT 505-510  
 (see also Task Global Table)  
 Transient Area 514  
 initialization coding generation  
 description 157  
 flowchart 297  
 segmented 514  
 storage map 501  
 object program listing  
 CLIST option 27  
 phase 6 output 141  
 OBJECT-COMPUTER paragraph 57  
 OBJSUB table, format 362  
 OBODOTAB table  
 built by phase 25 87,85  
 format 457  
 OBODOTBN cell (COMMON)  
 description 330  
 obtaining and printing error messages 50  
 OCCURS DEPENDING ON clause (see Q-routines;  
 OBODOTAB table)  
 OCCTBL table  
 format 363  
 input to phase 25 85  
 output of phase 22 75  
 usage 313,314  
 ODOBLD routine  
 diagram of 87  
 flowchart 261  
 ODOCT counter 87  
 ODOCTR cell (COMMON) 326  
 ODOTBL table  
 diagram of 87  
 format 363-364  
 use in phase 25 87  
 OD2TBL table  
 Data Division processing 59  
 format 364  
 input to phase 22 75  
 phase 10 processing 59  
 OFLOTBL  
 description 190,191  
 format 364  
 ON routine 137  
 ON SIZE ERROR clause 113  
 ON statement 137  
 ON strings 137  
 ONCTL field (TGT)  
 counter used for 143  
 description 509  
 location 505  
 ON processing 137  
 phase 62 counter for 163  
 ONCTR cell (COMMON)  
 description 329  
 use in phase 51 137  
 use in phase 6 143  
 use in phase 62 163  
 OPEN verb, debug processing 136  
 operating system  
 compilation  
 abnormal termination 31,32  
 invocation 31  
 compiler, relationship to 19  
 data management (see input/output requests)

operating system (continued)  
 object module, relationship to (see  
 input/output requests; error handling  
 in compilation; object module)  
 returning control to 31,32  
 operation code element  
 format 434  
 phase 64 action 184  
 operators, hierarchy of  
 definition 543  
 description 111  
 OPPRO routine, diagram of 581  
 OPT option  
 assembler coding for 110  
 compiler characteristics for 19  
 description 26  
 PERFORM statement processing with 110  
 phase 02 processing for 55  
 phase 4 processing for 106  
 phase 50 processing for 132  
 phase 51 processing 135,140  
 phase 62 processing for 161-174  
 phase 63 processing for 175-178  
 phase 64 processing for 179-186  
 segmentation operations 44  
 text generation for 140  
 Optimization A-text  
 compiler overview 25  
 definition 544  
 formats 439  
 generation 127  
 input/output operations 37-43  
 literal processing 139  
 phase 50 127,132  
 phase 50 introduction 119  
 phase 51 introduction 133  
 phase producing 316  
 optimization information elements  
 phase 4 format 428  
 phase 50 format 436  
 phase 50 processing of 132  
 phase 62 processing 171-174  
 phase 63 processing of 172-174,176  
 processing for 172-174  
 optimizing assignments (registers 14 and  
 15) 170  
 Optimizing literals 165  
 optimizing register assignments 168-170  
 optimizing storage for the PGT 164  
 optimizing virtuals 166  
 OPTINSW cell (COMMON) 333  
 OPTINSW1 cell (COMMON) 333  
 OPTINSW2 cell (COMMON) 333  
 OPTINSW3 cell (COMMON) 333  
 OPTINSW4 cell (COMMON) 333  
 optional phase processing 36  
 OPTLSTR cell (COMMON) 339  
 OPTLVL cell (COMMON) 339  
 OPTSWV2 cell (COMMON) 339  
 options  
 (see also compilation directives)  
 ADV 29  
 APOST 28  
 BATCH 26  
 BUF 25  
 CDECK 29  
 CLIST 27  
 CMS interface 531  
 COUNT 29  
 CSYNTAX 26  
 DECK 28  
 DISK 531  
 DMAP 27  
 DUMP 29  
 DYNAM 28  
 ENDJOB 28-29  
 FDECK 29  
 FLAG 28  
 FLOW 26  
 LANGLVL 29  
 LIB 25-26  
 LINECNT 28  
 LOAD 26  
 LCOL 29  
 LSTCOMP 29  
 LSTONLY 29  
 LVL 28  
 L120 29  
 L132 29  
 NAME 26  
 NUM 26  
 OPT 26  
 phase 62 output 161  
 PMAP 27  
 PRINT 531  
 QUOTE 28  
 RESIDENT 28  
 SEQ 28  
 SIZE 25  
 SOURCE 25  
 SPACE 28  
 STATE 26-27  
 SUPMAP 27  
 SXREF 27  
 SYMDMP 27  
 SYNTAX 26  
 SYST 28  
 SYSx 28  
 TERM 26  
 TEST 27  
 TRUNC 27-28  
 VBREF 29  
 VBSUM 29  
 VERB 28  
 XREF 27  
 ZWB 27  
 OPTSCN routine, function 539  
 out-of-line procedure, definition 544  
 OUTLRECL cell (COMMON) 339  
 output listing, suppression of 142  
 OU6REC work area, use in phase 64 179  
 OVERFLOW cell (TGT)  
 allocation (phase 62) 167  
 description 508  
 location 505  
 OVERFLOW cell (PGT) 511  
 OVERFLOW record  
 definition 544  
 description 190,191  
 PAGE clause, RDSCAN routine processing 62  
 PAGE-COUNTER data-name 522  
 PARAM field (TGT)  
 description 509-510

location 505  
 phase 6 processing 143  
 phase 62 counter for 164  
 parametric Report Writer routines 520,521  
 parentheses, Procedure IC-text  
 P0 format 412  
 P1 format 421  
 PARMAX cell (COMMON)  
   description 330  
   use in phase 6 143  
   use in phase 62 164  
 PARTBL table 193  
 PDATEX routine (phase 6) flowchart 294  
 PDATEX routine (phase 64) flowchart 294  
 PDSCN routine  
   description 66  
   flowchart 232  
 PERFORM cells (see also PFMSAV field (TGT)) 329  
 PERFORM statement, phase 4  
   processing 109-111  
 PF routine (see PGF-ROUT routine)  
 PFINDL routine 113  
 PFMCTL field (TGT)  
   counter in COMMON 329  
   description 509  
   location 505  
   phase 6 processing 143  
   phase 62 counter for 163  
 PFMCTR cell (COMMON)  
   description 329  
   use in phase 4 109  
   use in phase 6 143  
   use in phase 62 163  
 PFMSAV field (TGT)  
   counter in COMMON 329  
   description 509  
   location 505  
   PERFORM statement processing 109  
   phase 6 processing 143  
   phase 62 counter for 164  
 PFMSAV number 109  
 PFMIBL table  
   format 364-365  
   PERFORM statement processing 109  
 PGF-ROUT routine  
   description 521  
   GN number 528  
 PGH-ROUT routine 521  
 PGNARTN routine 165  
 PGT (see Program Global Table)  
 PGT-VN TABLE field (TGT)  
   description 507  
   location 505  
 PH routine (see PGH-ROUT routine)  
 phase, definition 544  
 phase, optional 36  
 phase 00  
   CSECT names 317  
   description 31-45  
   entry point  
     COS 35,44  
     (see also COMMON)  
     START 31  
   flowcharts 197-308  
   function 21  
   input/output requests 33,34  
   internal cells  
     LINKCNT 31,487  
     LINKNAME 487  
     SEGSAVE 44  
   microfiche directory 317  
   register usage 471  
   SEGIBL table 44  
 phase 01  
   CSECT names 317  
   description 46  
   flowchart 206  
   function 22  
   microfiche directory 317  
   register usage 471  
 phase 02  
   CSECT names 317  
   description 47-49  
   flowchart 207  
   function 22  
   microfiche directory 317  
   register usage 472  
 phase 03  
   CSECT names 317  
   description 50  
   diagram 597  
   flowchart 208  
   function 22  
   microfiche directory 317  
   register usage 472  
   returning control to phase 00 50  
 phase 04  
   description 51  
   diagram 561  
   entry point 317  
   flowcharts 209-212  
   function 22  
   input-output requests 51  
   microfiche directory 317  
   register usage 472  
   texts produced 316  
 phase 05  
   CSECT names 317  
   description 52-53  
   flowcharts 213-215  
   function 22  
   input/output requests 37  
   microfiche directory 317  
   register usage 473  
 phase 06  
   CSECT names 317  
   description 54  
   flowcharts 216-219  
   function 22  
   input/output requests 37  
   microfiche directory 317  
   register usage 473  
 phase 08  
   CSECT names 317  
   description 55  
   flowcharts 220-221  
   function 22  
   input/output requests 37  
   microfiche directory 317  
   register usage 473  
 phase 1B  
   CSECT names 318  
   description 66-70  
   entry point 318  
   flowchart 232

phase 1B (continued)  
   function 22-23  
   input/output requests 37  
   internal cell SEGSW 69  
   microfiche directory 318  
   register usage 475  
   table usage 313  
   texts produced 316  
   TIB usage 315  
 phase 3  
   CSECT names 318  
   description 88-98  
   entry points 318  
   flowcharts 251,253  
   function 23  
   input/output requests 38  
   microfiche directory 318  
   register usage 477  
   table usage 313  
   texts produced 316  
   TIB usage 315  
 phase 4  
   CSECT names 319  
   description 104-117  
   entry points 319  
   flowcharts 259-261  
   function 23  
   input/output requests 38  
   microfiche directory 319  
   optimization information elements 428  
   register usage 477  
   texts produced 316  
   table usage 313  
   TIB usage 315  
 phase 6  
   CSECT names 319  
   description 141-160  
   entry points 319  
   flowcharts 276-281  
   function 24  
   input/output requests 31  
   microfiche directory 319  
   OU6REC internal cell 151  
   register usage 479  
   table usage 314  
   texts produced 316  
   TIB usage 315  
 phase 6A  
   CSECT names 320  
   flowcharts 300-302  
   description 190,191  
   function 24-25  
   input/output requests 42  
   microfiche directory 320  
   register usage 481  
   table usage 314  
   TIB usage 315  
 phase 10  
   communication section processing 60  
   CSECT names 317  
   description 56-60  
   entry point 317  
   flowcharts 222-225  
   function 22  
   input/output requests 37  
   internal cell CURGCN 56  
   microfiche directory 317  
   register usage 474  
   table usage 313  
   texts produced 316  
   TIB usage 315  
   work area ICTEXT 59  
 phase 12  
   CSECT names 317  
   description 61-65  
   flowcharts 226-231  
   function 22  
   generating error messages 64  
   input/output flow 63  
   input/output requests 37  
   microfiche directory 317  
   register usage 474-475  
   table usage 313  
   texts produced 316  
   TIB usage 315  
 phase 20  
   communication section processing 73  
   CSECT names 318  
   description 71-73  
   entry point 318  
   function 23  
   flowcharts 233-236  
   input/output requests 37  
   DCBEXLST cell 80  
   microfiche directory 318  
   register usage 475  
   table usage 313  
   texts produced 316  
   TIB usage 315  
 phase 21  
   CSECT names 318  
   description 79-84  
   flowchart 245-246  
   function 23  
   input/output requests 38  
   microfiche directory 318  
   register usage 476  
   table usage 313  
   texts produced 316  
   TIB usage 315  
 phase 22  
   CSECT names 318  
   description 74-78  
   flowcharts 237-244  
   function 23  
   input/output requests 38  
   microfiche directory 318  
   register usage 476  
   table usage 313  
   texts produced 316  
   TIB usage 315  
 phase 25  
   CSECT names 318  
   description 85-87  
   diagram of operations 577  
   flowchart 247-250  
   function 23  
   input to 87  
   microfiche directory 318  
   operations 87  
   operations diagram 577  
   register usage 476  
   table usage 313  
   TIB usage 315  
 phase 35  
   CSECT names 318



- description 99-103
- entry point 317
- flowcharts 254-258
- function 23
- microfiche directory 318
- register usage 477
- routines 100-102
- table handling 100
- table usage 313
- TIB usage 315
- phase 45
  - CSECT names 319
  - description 118
  - flowcharts 262-264
  - function 23
  - input/output requests 39
  - microfiche directory 319
  - register usage 478
  - table usage 313
  - texts produced 316
  - TIB usage 315
- phase 50
  - CSECT names 319
  - description 119-132
  - entry points 319
  - flowcharts 265-270
  - function 24
  - input/output requests 40
  - microfiche directory 319
  - optimization information elements 436
  - register usage 478
  - table usage 313
  - texts produced 316
  - TIB usage 315
- phase 51
  - CSECT names 319
  - description 133-140
  - entry points 319
  - flowcharts 271-275
  - function 24
  - input/output requests 40
  - microfiche directory 319
  - register usage 478
  - table usage 313
  - texts produced 316
  - TIB usage 315
- phase 62
  - CSECT names 319
  - description 161-174
  - diagram of 581
  - flowcharts 282-286
  - function 24,161
  - input/output requests 41
  - microfiche directory 319
  - operations 581
  - register usage 479
  - table usage 313
  - TIB usage 315
- phase 63
  - CSECT names 320
  - description 175-178
  - diagram 583
  - flowchart 287-292
  - function 24
  - input/output requests 41
  - microfiche directory 320
  - register usage 480
  - table usage 313
- texts produced 316
- TIB usage 315
- phase 64
  - CSECT names 320
  - description 179-186
  - flowchart 293-297
  - function 24,179
  - input/output requests 42
  - microfiche directory 320
  - output 179
  - register usage 480
  - table usage 313
  - texts produced 316
  - TIB usage 315
- phase 65
  - CSECT names 320
  - description 187-189
  - diagram 585
  - flowcharts 298-299
  - function 24,187
  - input/output requests 42
  - microfiche directory 320
  - register usage 480
  - table usage 314
  - TIB usage 315
- phase 70
  - CSECT names 320
  - description 192-194
  - entry points 320
  - flowchart 303
  - function 25
  - input/output requests 43
  - microfiche directory 320
  - register usage 481
  - table usage 314
  - TIB usage 315
  - XU6REC internal cell 193
- phase 71
  - CSECT names 320
  - description 192-194
  - microfiche directory 320
- phase 72
  - CSECT names 320
  - description 192-194
  - microfiche directory 320
- phase 80
  - CSECT names 320
  - description 195-196
  - flowcharts 304-308
  - function 25
  - input/output requests 43
  - microfiche directory 320
  - register usage 482
- PHASEND routine 86
- PHAS63 routine 175,177
- PHCTRL routine
  - flowchart 253
  - phase 3 control 88-89
  - SEARCH string processing 93
- PHINIT routine
  - description 98
  - Glossary building 88,597
- PHNESS table, use in phase 70 192
- PHxERR tables 193,194
- PHZERR cell (COMMON) 338
- PHZSW cell (COMMON)
  - description 333
  - use in phase 02 47,48

PHZSW1 cell (COMMON)  
 description 333  
 use in phase 02 47,48  
 use in phase 3 (SYM bit) 88  
 use in phase 6A 190

PHZSW2 cell (COMMON)  
 description 334  
 use in phase 02 47,48

PHZSW3 cell (COMMON)  
 description 334  
 statistical information 48  
 syntax-checking function 45  
 use in phase 02 47,48

PHZSW4 cell (COMMON) 334

PHOSW cell (COMMON) 339

PH1BYTE cell (COMMON)  
 Configuration Section processing 57  
 description 335

PH1ERR table, use in phase 70 193,194

PH2ERR table, use in phase 71 192

PH25SW cell (COMMON) 338

PH3ERR table, use in phase 72 192

PH4ERR table, use in phase 72 192

PH45CTL routine 118

PH45BIT bit (COMMON), set in phase 4 104

PH5CTL routine  
 debug option processing 135  
 flowchart 266  
 handling phase 51 verb strings 132  
 ON processing 137  
 verb string processing 119,120

PH5ERR table, use in phase 72 192

PH6 routine (phase 6) flowchart 277

PH6 routine (phase 62) flowchart 283

PH6ERR cell (COMMON)  
 description 329  
 phase 70 processing 193

PH6ERR table, use in phase 72 192

PH62 routine 581

PH65 routine 585

PIOTBL table  
 format 365  
 input to phase 12 63  
 output of phase 12 63  
 phase 1B processing 66  
 RERUN clause processing 58-59  
 use in phase 21 79  
 verb processing 69,70

PLSCALL routine 205

PMAP option  
 description 27  
 phase 02 parameters 47  
 phase 62 output 161  
 phase 64 processing for 179  
 register assignments 483  
 switch for 333

PN (source program procedure-name)  
 allocation (phase 62) 167  
 DEF-text  
 format 443  
 phase 64 action 179  
 Procedure A-text format 434  
 Procedure IC-text formats 411,419  
 description  
 definition 544  
 dictionary entry formats 446  
 phase 1B 66,67  
 phase 50 130  
 phase 51 134  
 equate strings  
 description 144  
 Optimization A-text format 439  
 field (PGT) 511  
 number  
 phase 1B 66,67  
 phase 4 104  
 phase 6 150  
 optimizing 146  
 reference element  
 phase 64 action 184  
 Procedure A-text format 434  
 Procedure IC-text formats 411,419

PNATBL table  
 format 366  
 phase 62 processing 165

PNCHSW routine  
 description 189  
 diagram 585

PNCHSW switch, use in phase 65 189

PNCTR cell (COMMON)  
 description 325  
 use in phase 1B 66  
 use in phase 51 135

PNDEF routine  
 diagram of 583  
 flowchart 290

PNDEFRTN routine (phase 62) 581

PNDEFRTN routine 102

PNEQR routine 146

PNFWDBTB table  
 building of 171  
 format 366

PNLABTBL table  
 building of 171  
 format 366  
 use in phase 63 175  
 use of ACCUMCTR 171

PNLBDBTB table, format 367

PNOUNT table  
 description 111,112  
 format 367

PNQTBL table  
 description 67,68  
 format 367

PNTABL table  
 description 67,68  
 format 368

PNTBL table  
 format 368  
 optimizing PNs 583  
 PN allocation 149-150  
 Procedure A-text elements 152,153

PNUREF element, optimization A-text  
 format 440

PNUTBL table  
 building of 135  
 format 369  
 optimizing PNs 146  
 phase 6 introduction 141

PRBLDISP cell (COMMON)  
 description 325  
 phase 64 processing 186

PRBLNUM cell (COMMON) 330

PRBL1 CELL PTR field (TGT)  
   description 508  
   location 505  
 PRFORM routine  
   description 109  
   flowchart 261  
 PRFTWO routine (phase 6)  
   flowchart 278  
   diagram of 581  
 PRFTWO routine (phase 62) flowchart 284  
 PRIME routine  
   called by MOVDIC 499  
   description 498-499  
 PRINT option 531  
 PRINT-SWITCH data-names 522  
 PRINTBUF cell (COMMON) 338  
 priority, definition 544  
 priority elements, debug-text format 444  
 priority numbers in segmentation  
   checking in phase 1B 69  
   Configuration Section 57  
   procedure A-text processing 151-152,155  
 PROCBL counter  
   use in branch instruction  
   processing 174  
   use in building PNLABTBL and  
   GNLABTBL 171  
   use in phase 62 169  
 PROCCTR cell (COMMON) 330  
 procedure, in-line, definition 544  
 procedure, out-of-line, definition 544  
 Procedure A-text  
   data sets used for 37-43  
   definition 544  
   formats 434-438  
   use in procedure block  
   allocation 168,169  
 procedure A1-text  
   definition 545  
   formats 441-442  
   phase 64 action 179,183-186  
   phase 64 processing of 179  
   producing of 175  
 procedure base register for GNs element  
   phase 64 action 182  
   procedure A1-text format 441  
 procedure base register for PNs element  
   phase 64 action 182  
   procedure A1-text format 441  
 Procedure Block  
   allocation 168,169  
   definition 545  
   phase 63 processing for 175  
   segmentation processing 171  
 PROCEDURE BLOCK field (PGT)  
   description 512  
   location 510  
   overflow allocation 166  
 procedure block number element, phase 64  
   action 183  
 Procedure Division  
   compiler design 20  
   overview 21-25  
   phase 1B processing 66  
   procedure-name processing 104  
 Procedure field (object module)  
   description 512  
   location 510  
 Procedure IC-text  
   data sets used for 37-43  
   definition 545  
   description 21  
   formats 411-432  
 procedure-name DEF-text element, phase 64  
   action 182  
   procedure-name reference, optimization  
   A-text format 440  
   procedure-name, compiler generated (see GN)  
   procedure-name, source program (see PN)  
   procedure-name, variable (see VN)  
   procedure P1A-text 424  
 PROCESLD routine 86  
 processing for branch instructions  
   phase 62 174  
   phase 63 175  
 processing for optimization information  
   elements 176  
 PROCINDX table  
   building of 188  
   format  
   compiler 369  
   debug data set 466  
 PROCNOTE save area, use in phase 65 188  
 PROCENM routine 86  
 PROCTAB table  
   building of 187  
   format 465  
   location in object module 501  
   object module 514  
 PROC01 routine  
   description 64  
   flowchart 228  
 PROC02 routine  
   description 64  
   flowchart 229  
 producing a storage dump 29,470  
 producing file section entries 71  
 producing incomplete Data A-text 72  
 producing the report writer subprogram 62  
 PROGID cell (COMMON)  
   description 325  
   use in phase 10 57  
 program break 134  
   (see also critical program breaks)  
 program collating sequence, phase 05 57  
 Program Global Table (PGT)  
   base registers for 182  
   COMMON, relationship to 324  
   counters used for 146  
   definition 545  
   description 510-512  
   fields  
   COUNT LINKAGE AREA 511  
   DCBADR 511  
   DEBUG LINKAGE AREA 510  
   DISPLAY LITERAL 512  
   GN 511  
   LITERAL 511-512  
   locations 510  
   OVERFLOW 511  
   PN 511  
   PROCEDURE BLOCK 512  
   TEST LINKAGE AREA 511  
   VIRTUAL 511  
   VIRTUAL EBCDIC NAMES 511  
   VNI 511

Program Global Table (PGT) (continued)  
   phase 6 introduction 141  
   phase 6 storage allocation 148-151  
   phase 62 storage allocation for 162-166  
   phase 64 processing for 179  
 progress messages  
   definition 545  
   interphase processing 34  
   SYSTEM data set 44  
   TERM option 26  
 PROGSUM table  
   building of 188  
   format 456  
 PROGSW cell (COMMON) 338  
 PSHTBL table  
   description 113  
   format 370  
 PSIGNT table  
   description 111-113  
   format 370  
 PSVCTR cell (COMMON)  
   description 329  
   use in phase 6 143  
   use in phase 62 164  
 PTRFLS table  
   description 113  
   format 370  
 PTYNO cell (COMMON) 327  
 PUT routine (phase 6) 151  
 PUTDEF routine  
   flowchart 274  
   PN definition 134  
 P0-text (see procedure IC-text)  
 P1-text (see procedure IC-text)  
 P1BTBL table  
   description 66  
   format 371  
   input to phase 12 63  
   output of phase 12 63  
   use in phase 10 61  
 P1TEXT table  
   description 100  
   format 372  
 P2-text (see procedure IC-text)  
  
 Q-routine, definition 545  
 Q-routine generation 77-78  
 Q-routine identification elements  
   A-text processing for 77  
   phase 64 action 182  
 QALTL table 372  
 QBEGIN routine 175  
 QBUILD routine 78  
 QFILE table  
   attribute replacement 95  
   format 372-373  
   phase 21 processing 79  
   use in phase 3 95  
 QGNTBL table  
   building of 177  
   format 373  
   use in phase 63 175  
   use in phase 64 182  
 QITBL table  
   diagram of 87  
   format 373  
   input to phase 25 85  
  
 QISAM access method 80-83  
 QLTLBL table  
   format 374  
   input to phase 12 63  
 QNMTBL table  
   description 57-59  
   format 374  
 QRTN table  
   diagram of 87  
   format 374-375  
   input to phase 25 85  
 QSAM access method 80-83  
 QSBL table 375  
 QTBL table  
   format 375  
   Q-routine identification element 160  
   use in phase 64 182  
 QUALIF routine 95  
 qualified name, definition 545  
 qualifying EBCDIC name, Procedure IC-text  
   format 411  
 QUOTE option 28  
 QVAR table  
   dictionary attributes 95  
   format 375-376  
   phase 22 processing 75  
 QVARBD routine 78  
  
 RCDTBL table  
   Data Division processing 59  
   format 376  
   input to phase 12 63  
   output of phase 12 63  
 RC4 routine flowchart 296  
 RC8C routine flowchart 296  
 RD dictionary entries  
   Data IC-text format 402  
   description 59  
   format 449  
   preprocessing 76  
 RDFSTK table  
   format 376  
   use in phase 22 76,77  
 RDF2 routine  
   description 188  
   diagram 585  
 RDSCAN routine  
   description 62  
   flowchart 227  
 RDSYN routine 77  
 RD001 routine flowchart 296  
 READ routine  
   description 33  
   flowchart 200  
 READ statement 89  
 READ verb 136  
 READ verb, debug processing 135  
 READFN routine 89  
 READF2 routine (phase 6) 146  
 READF2 routine (phase 62) diagram 581  
 READF4 routine  
   description 74  
   flowchart 243  
 READLIB routine, flowchart 204  
 RECEIVE verb  
   phase 50 processing for 129  
   phase 51 processing 136

Record Description dictionary entries 60  
 (see also RD dictionary entries)  
 RECORD KEY clause 64  
 REDEF routine 76-77  
 REF-text  
     data sets used for 37-43  
     definition 545  
     format 443  
     phase 6A 190,191  
 REFLD11 routine diagram 585  
 REDEFINES clause, phase 22 processing  
     of 62,76-77  
 REGCOMAD routine, diagram 585  
 register assignment  
     CMS interface routine 539  
     description 155,156  
     NOOPT option 483  
     OPT option 483  
     optimization of 168-170  
     phase 51 verb processing 135  
 register specification element  
     format 435  
     phase 64 action 185  
 register usage  
     compilation 471-482  
     execution-time 155,156  
     saving registers 487  
     work registers 135  
 register 14, optimizing assignment of 170  
 register 15, optimizing assignment of 170  
 REGLIST routine 581  
 REGMTX table 151-156  
 REGMV1 routine  
     diagram 581  
     use in register assignments 169  
 RELADD cell (COMMON) 325  
 relational codes, Procedure IC-text  
     format 412  
     P1 format 421  
     P2 format 428  
 relative addresses element  
     format 435  
     phase 64 action 185  
 RELEASE verb, debug processing 135  
 RELLOC cell (COMMON) 329  
 relocation dictionary, VIRTUAL  
     allocation 149  
 RELSPACA cell (COMMON) 337  
 RENAMES, dictionary entries 74  
 RENAMS routine 74  
 RENAMTB table  
     format 376-377  
     input to phase 25 85  
 REPORT clause, phase 10 processing 61  
 REPORT routine flowchart 235  
 REPORT SAVE field (TGT), phase 62 counter  
     for 164  
 Report Section, phase 12 processing for 61  
 report section dictionary entries, phase 22  
     processing of 77  
 report section header, use in phase 10 61  
 Report Writer  
     data-names 522,523  
     description 515-528  
     fixed routines 515  
     group routines 521,522  
     parametric routines 520,521  
     routines generated for  
         subprogram 515-522  
         source statements, compiler response  
         to 516,517  
         subprogram  
             elements of 515-523  
             logic of 516-519  
             verbs 523,524  
     Report Writer Subprogram (RWS), phase 12  
         processing for 61  
 REPORT-CALL verb 523  
 REPORT-ORIGIN verb  
     created by phase 1B 61  
     description 523  
     OPT processing for 176  
 REPORT-REORIGIN verb  
     created by phase 1B 61  
     description 523  
 REPORT-RETURN verb 523  
 REPORT-SAVE verb 523  
 REPTAB table  
     format 377  
     input to phase 12 63  
     output of phase 12 63  
 RERUN bit  
     Input/Output verb processing 69  
     RERUN clause processing 58-59  
 RERUN clause 58-59  
 RERUN verb, phase 51 processing 136  
 RESERVE element  
     function 120  
     use in phase 62 169  
 RESIDENT option  
     allocation of virtuals for 149  
     description 28  
     phase 02 processing 47  
     phase 51 processing for 139  
     phase 6 processing for 149  
     phase 62 virtual EBCDIC names allocation  
         for 167  
     virtual allocation for 167  
 RET CODE field (TGT)  
     description 507  
     location 505  
 RET-ROUT routine  
     description 520  
     generation of 65  
 RETURN macro instruction 32  
 RETURN verb, debug processing 135  
 REWRITE verb, debug processing 135  
 RGNCTR cell (COMMON)  
     description 327  
     use in phase 62 167  
 RLD-text  
     definition 545  
     description 156  
 RLDSORT routine, diagram 583  
 RLDTBL table  
     completion by phase 64 179  
     format 377,378  
     making entries in 177  
     phase 6 introduction 141  
     phase 64 action for 181  
     phase 64 table processing 183  
     processing Data A-text, E-text,  
         DEF-text 159-160

RLDTBL table (continued)  
   processing Procedure A-text  
   elements 152  
   use in phase 63 176  
 RLS-ROUT routine 521  
 RNMTBL table  
   format 378,379  
   output of phase 12 63  
   phase 1B introduction 66  
   use in phase 22 77  
 ROL-ROUT routine  
   generation of 64  
   logic of report writer subprogram 517  
 ROLTEL table 379  
 root segment, definition 545  
 ROUNDED clause 113  
 ROUTBL table  
   description 66  
   format 380  
   output of phase 12 63  
 routines  
   ACCESS routines (see ACCESS routines)  
   interlude 37-43  
   Report Writer 515-522  
   SYNAD 32,34  
   verb analyzer (see verb analyzer  
   routines)  
 RPF-ROUT routine  
   description 521  
   GN number 528  
 RPH-ROUT routine 521  
 RPNCNTR cell (COMMON)  
   description 328  
   use in phase 62 167  
 RPT field (object module)  
   description 512  
   location 501  
 RPT-ORIGIN elements  
   phase 63 processing of 176  
   Procedure A-text format 436  
 RPT.LIN work area, phase 22 processing 62  
 RPT.RCD data-name 522  
 RPTSAV field (TGT)  
   description 510  
   location 505  
   processing for 146  
 RPTSAV cell (COMMON)  
   description 327  
   use in phase 62 164  
   use in phase 6 146  
 RSECT routine flowchart 241  
 RPTWTR bit (COMMON), use in phase 10 61  
 RST-ROUT routine  
   description 520  
   generation of 64  
 RUNTEL table  
   format 380-381  
   usage 313,314  
 RWRTBL table  
   description 66  
   format 381  
   input to phase 12 63  
   output of phase 12 63  
 R6400 routine diagram of 581  
  
 S.nnnn data-name 523  
 SAME AREA clause  
   buffer generation 83  
   I-O-Control paragraph processing 58-59  
   phase 21 processing 83  
 SAME RECORD AREA clause, phase 21  
   processing for 83  
 SAME RECORD clause 58  
 SAME routine 83  
 SAMER routine 83  
 SAMETB table  
   buffer generation 83  
   buffer processing 83  
   FD dictionary entries 80  
   format 381  
   usage 313,314  
   use in phase 21 83  
 SATBL table  
   description 58  
   format 382  
 SAVE AREA field (TGT)  
   description 505  
   location 505  
 SAVE AREA-2 field (TGT)  
   description 509  
   location 505  
   phase 6 processing 146  
   phase 62 counter for 164  
 SAVE AREA-3 field (TGT)  
   description 509  
   location 505  
   phase 6 processing 146  
   phase 62 counter for 164  
 SAV-ROUT routine  
   description 520  
   generation of 64  
   GN number 528  
 SAVE routine 581  
 SAVETBL table, use in phase 63 175  
 SA2CTR cell (COMMON)  
   description 327  
   use in phase 6 146  
   use in phase 62 164  
 SA3CTR cell (COMMON)  
   description 332  
   use in phase 6 146  
   use in phase 62 164  
 SBL  
   counter used for 143  
   definition 545  
 SBL field (TGT)  
   description 509  
   location 505  
   phase 62 counter for 163  
 SBLCTR cell (COMMON)  
   description 326  
   use in phase 6 143  
   use in phase 22 76  
   use in phase 62 163  
 SD dictionary entries  
   Data IC-text format 402  
   format 449  
   phase 21 processing of 80  
   preprocessing 76  
 SD entries 59  
 SD-text, definition 545  
 SDSIZ cell (COMMON) 328

SDTEXT routine  
   description 80  
   SD dictionary entries 80  
 SEARCH routine  
   determining uniqueness 95  
   phase 3 processing 89  
 SEARCH verb analyzer routine 113-114  
 secondary base locator, definition 545  
 section, definition 545  
 SEGBRK routine 177  
 SEGENTR routine 134  
 SEGINDX table  
   building of 188  
   format 382,466  
   object module 514  
 SEGLMT cell (COMMON)  
   description 328  
   use in phase 1B 69  
   use in phase 10 57  
 segment, definition 545  
 segment elements  
   phase 65 processing 188  
 SEGMENT-LIMIT clause, phase 63 processing  
   of 177  
 segmentation  
   control breaks 134  
   definition 545  
   elements, format 439  
   optimization with 171  
   phase 00 processing 44  
   phase 1B 69  
   phase 10 57  
   phase 51 134  
   phase 6 155  
   phase 63 processing for 177  
 segmentation call parameter element  
   phase 64 action 185  
   procedure A-text format 436  
 SEGNOTE routine 33  
 SEGPT routine  
   linkage codes to 33  
   segmentation operations 44  
 SEGPROC routine 155  
 SEGTL table  
   format 383  
   Procedure A-text processing 155  
   segmentation control breaks 134  
   segmentation operations 44  
 SELECT clause 58  
 SELSCN routine 49  
 SEND verb  
   phase 50 processing 132  
   phase 51 processing 137  
 SEQ option  
   description 28  
   phase 02 parameters 47  
 SEQERR cell (COMMON) 326  
 sequence option  
   description 28  
   phase 02 parameters 47  
   switch for 333  
 SETNAMS routine  
   description 86  
   flowchart 250  
 SETTBL table 383  
 severity of errors 31  
 SE6000 routine (phase 6) flowchart 279  
 SE6000 routine (phase 62)  
   diagram of 581  
   flowchart 285,286  
 SE6000 routine (phase 64) flowchart 295  
 SIZE ERROR clause 113  
 SIZE option  
   description 25  
   error handling 49  
   phase 02 parameters 47  
 SKIP routine 33  
 SMRCDTBL table  
   buffer generation 83  
   FD dictionary entries 80  
   format 384  
   usage 313,314  
   use in phase 21 83  
 SMSTBL table 384  
 SNF routine (phase 65), diagram 585  
 SNMTBL table 385  
 SORT CORE SIZE field (TGT)  
   description 507  
   location 505  
 Sort Description dictionary entries (see SD  
   dictionary entries)  
 Sort Description entries 59  
 SORT FILE SIZE field (TGT)  
   description 507  
   location 505  
 SORT-MESSAGE field (TGT)  
   description 507-508  
   location 505  
 SORT MODE SIZE field (TGT)  
   description 507  
   location 505  
 SORT RET field (TGT)  
   description 507  
   location 505  
 SORT SAVE field (TGT)  
   description 507  
   location 505  
 SORTXT routine  
   description 118  
   flowchart 264  
 source listing, generated by phase 12 65  
 source module  
   definition 545  
   description 20  
 SOURCE option  
   description 25  
   Identification Division 57  
   phase 02 parameters 47  
   switch for 333  
 source program errors (see error handling  
   in compilation)  
 source program procedure-name (see PN)  
 SPACE option  
   description 28  
   phase 02 parameters 47  
 SPACEX cell (COMMON) 336  
 SPACING cell (COMMON)  
   description 330  
   use in phase 00 48  
   use in phase 02 47  
 SPCRTS routine flowchart 231  
 special names, Procedure IC-text  
   P1 format 412  
   P2 format 428  
 SPECIAL-NAMES paragraph 57

special phase 6 elements, Procedure A-text  
 format 435

SPNTBL table  
 Configuration Section processing 57  
 format 385  
 input to phase 12 63  
 output of phase 12 63  
 verb processing 69

SRATBL table  
 description 58  
 format 386

SRCHKY table  
 format 386  
 use in phase 22 75

SRCTBL table 386

SSCIN table  
 format 387  
 usage 313,314  
 use in phase 45 118

SSCOUT table  
 format 387  
 usage 313,314  
 use in phase 45 118

SSCRPT routine 123

SSDELIM table  
 format 388  
 usage 313,314  
 use in phase 45 118

STAESW cell (COMMON) 336

standard data-name references, Procedure  
 IC-text  
 format 412  
 P1 format 422  
 P2 format 428

START entry point 31

STATE option  
 bit in COMMON 334  
 debug text construction for 176  
 description 26-27  
 phase 02 parameters 47  
 phase 10 processing for 56,57  
 phase 50 processing 120,121  
 phase 51 processing 135  
 phase 6 output 142  
 phase 62 output 162  
 phase 63 processing for 176  
 phase 05 processing for 187,188  
 Procedure A-text processing 151  
 segmentation processing for 178  
 TGT allocation for 164

statement number option (see STATE option)

STATIC routine 499

STOP RUN statement, ENDJOB option  
 for 28-29

storage requirements for compiler 29-30

STRING table  
 COMPUTE statement processing 112  
 format 388  
 MOVE statement processing 104  
 SEARCH statement processing 113-114

STRING verb, phase 51 processing 137

strings  
 ADD 124  
 CALL 105  
 EQUATE 109  
 EVAL 113  
 GO 105,106  
 IF 113

MOVE 104

SUBSCRIPT 120,121

STSRCH routine  
 PO-text translation 89  
 SEARCH format-2 processing 93

SUBADR field (TGT)  
 counter in COMMON 329  
 counter used for 143  
 description 509  
 location 505  
 ON processing 137  
 phase 62 counter for 163

SUBCOM PTR field (TGT)  
 description 507  
 location 505  
 use in phase 51 137  
 use in phase 62 163

SUBCTR cell (COMMON) 329

subject hierarchy, definition 545

subscript save cell  
 phase 50 126  
 phase 6 137

SUBSCRIPT string  
 MOVE statement processing 104  
 resolving references 120-122

subscripted references  
 data-name  
 optimizing 121,122  
 Procedure IC-text format 426  
 resolving 120-122

literal  
 optimizing 122,123  
 resolving 121,122

SUMTBL table 388-389

SUPMAP option  
 description 27  
 phase 02 parameters 47  
 phase 6 processing 142  
 phase 62 output 162  
 switch for 333

suppression of output listing 142

SWITCH cell (COMMON)  
 description 334  
 syntax-checking function 45  
 use in phase 10 (RERUNN bit) 58-59  
 use in phase 6 142  
 use in phase 70 159

SWITCH field (TGT)  
 description 505-507  
 location 505

SWITCH1 cell (COMMON) 337

SWITCH2 cell (COMMON) 336

SWITV2 cell (COMMON) 330

SXREF option  
 description 27  
 interphase processing for 31  
 phase 02 parameters 47  
 phase 6 output 141,142  
 phase 6A processing 190,191  
 phase 62 output 161  
 phase 64 processing for 179  
 switch for 333

SYAA routine  
 error recovery response 470  
 phase input/output requests 34

SYAB routine  
 error recovery response 470  
 phase input/output requests 34



SYAD routine  
 error recovery response 470  
 phase input/output requests 34  
 SYMDICT DSECT, function 86  
 SYMDMP option  
 debug linkage area for 166  
 debug text construction for 176  
 description 27  
 phase 02 processing for 47  
 phase 10 processing for 56,57  
 phase 22 processing 78  
 phase 25 processing for 85-87  
 phase 50 processing 120,121  
 phase 51 processing 135  
 phase 63 processing for 176  
 phase 65 processing for 187,188  
 segmentation processing for 177-178  
 TGT allocation for 164  
 SYMSK cell (COMMON) 333  
 SYMSK1 cell (COMMON) 334  
 SYMSK2 cell (COMMON) 334  
 SYMSK3 cell (COMMON) 334  
 SYMWRITE routine, diagram 585  
 SYNAD exit, phase 03 processing for 50  
 SYNAD routines  
 error recovery 470  
 phase input/output requests 34  
 SYNADRO1 cell (COMMON)  
 description 332  
 phase 03 processing 50  
 syntactic and reference markers  
 IPTTEXT format  
 one byte 400  
 two byte 400  
 syntax analysis, phase 05 52-53  
 SYNTAX option  
 (see also CSYNTAX option)  
 description 26  
 phase 00 processing for 45  
 phase 02 processing for 47  
 phase 21 processing 79  
 phase 3 processing for 96  
 phase 4 processing 116  
 phase 50 processing 119  
 phase 51 processing 133  
 E-text processing 192  
 syntax-checking compilations 45  
 syntax checking, compiler output for 20  
 SYSIN data set  
 buffer size determination 47  
 input to compiler 20  
 SYSLIB data set, CMS FILEDEF for 532  
 SYSLIN data set  
 compiler active 37-43  
 output of compiler 20  
 SYSOUT DDNAME field (TGT)  
 description 508  
 location 505  
 SYSPRINT data set  
 compiler activity 37-42  
 error messages to 44  
 Identification Division 57  
 output of compiler 26  
 phase 70 output 25  
 SYSPUNCH data set  
 CMS FILEDEF for 532  
 CMS processing of 533  
 compiler activity 37-43  
 output of compiler 20  
 SYST option 28  
 SYSTDD cell (COMMON) 338  
 SYSTX cell (COMMON) 328  
 System/370  
 phase 10 57  
 phase 50 127  
 phase 51 140  
 SYSTEM data set  
 CMS FILEDEF for 532  
 compiler activity 37-43  
 progress and error messages to 44  
 output of compiler 20  
 phase 70 output 25  
 SYSUT1 data set  
 CMS FILEDEF for 532  
 compiler activity 37-43  
 general 21  
 SYSUT2 data set  
 CMS FILEDEF for 532  
 compiler activity 37-43  
 general 21  
 STATE option 26-27  
 SYSUT3 data set  
 CMS FILEDEF for 532  
 compiler activity 37-43  
 Data Division processing 59  
 general 21  
 SYSUT4 data set  
 CMS FILEDEF for 532  
 compiler activity 37-43  
 contents in phase 64 180  
 general 21  
 SYSUT5 data set  
 CMS FILEDEF for 532  
 compiler activity 37-43  
 general 21  
 SYSUT6 data set  
 compiler activity 37-43  
 FIPS processing 28  
 general 21  
 LVL option 20,28  
 SYSx option  
 description 28  
 phase 02 processing for 47  
 SYS2 routine, diagram of 583  
 TA LENGTH field (TGT)  
 description 508  
 location 505  
 Table Area Management Executive Routines  
 (see TAMER)  
 Table Area Management Maps (TAMM) 497  
 table, definition 545  
 table entry definition 543  
 table formats 340-398  
 table handling (see TAMER)  
 Table Information Blocks (TIB)  
 COMMON cell 325  
 definition 546  
 TAMER Control field 496-497  
 usage 315  
 table locator, definition 545  
 Table usage 313,314  
 tables, locating 313,314  
 tables used by compiler, formats 340-398

TABREL routine 499-500  
TALLY field (TGT)  
    description 507  
    location 505  
TAMEIN routine 498  
TAMEOP routine 500  
TAMER  
    definition 546  
    description 492-500  
    initialization 498  
    interphase processing 31  
    register usage 151  
    routines 498-500  
    space, definition 546  
    storage allocation 498  
TAMM (Table Area Management Maps) 497  
TAMNAD cell (COMMON) 325  
Task Global Table (TGT)  
    COMMON, relationship to 324  
    definition 546  
    base registers for 182  
    fields  
        A (INIT1) 507  
        BL 508  
        BLL 509  
        CHECKPT CTR 510  
        COBOL ID 508  
        COBOL INDICATOR 507  
        COMPILED POINTER 508  
        COUNT CHAIN ADDRESS 508  
        COUNT TABLE ADDRESS 508  
        DBG R11SAVE 508  
        DDBG R14SAVE 507  
        DEBUG BLL 508  
        DEBUG CARD 508  
        DEBUGGING 508  
        DEBUG MAX 508  
        DEBUG PTR 508  
        DEBUG TABLE 510  
        DEBUG TABLE PTR 507  
        DEBUG TRANSFER 508  
        DEBUG VLC 508  
        DECBADR 508  
        description 505-510  
        ENTRY-SAVE 507  
        FIB 508  
        INDEX 509  
        LABEL RET 507  
        LENGTH OF VN TBL 507  
        locations 505  
        ONCTL 509  
        OVERFLOW 508  
        PARAM 509-510  
        PCS LIT PTR 508  
        PFMCTL 509  
        PFMSAV 509  
        PGT-VN TBL 507  
        PRBL1 CELL PTR 508  
        RET CODE 507  
        RPTSAV 510  
        SAVE AREA 505  
        SAVE AREA-2 509  
        SAVE AREA-3 509  
        SBL 509  
        SORT CORE SIZE 507  
        SORT FILE SIZE 507  
        SORT-MESSAGE 507-508  
        SORT MODE SIZE 507  
        SORT RET 507  
        SORT SAVE 507  
        SUBADR 509  
        SUBCOM PTR 507  
        SWITCH 505-507  
        SYSOUT DDNAME 507  
        TA LENGTH 507  
        TALLY 507  
        TEMP STORAGE 508-509  
        TEMP STORAGE-2 509  
        TEMP STORAGE-3 509  
        TEMP STORAGE-4 509  
        TGT-VN TBL 507  
        VLC 509  
        VN 509  
        WORKING CELLS 507  
        XSA 509  
        XSASW 509  
        phase 64 processing for 179  
        storage allocation 143  
TBGETSPC routine 499  
TBLRPT table 61  
TBREADIC routine 500  
TBSPILL routine 500  
TBWRITE routine 500  
TEMP STORAGE field (TGT)  
    description 508-509  
    location 505  
    phase 6 counter used for 143  
    phase 6 processing 143  
    phase 62 counter used for 163  
    use in phase 50 126  
TEMP STORAGE-2 field (TGT)  
    description 509  
    location 505  
    phase 6 counter used for 143  
    phase 6 processing 143  
    phase 62 counter for 163  
TEMP STORAGE-3 field (TGT)  
    description 509  
    location 505  
    phase 6 counter used for 143  
    phase 6 processing 143  
    phase 62 counter for 163  
TEMP STORAGE-4 field (TGT)  
    description 509  
    location 505  
    phase 6 counter used for 143  
    phase 6 processing 143  
    phase 62 counter for 163  
temporary result, definition 546  
temporary result references, Procedure  
    IC-text format 429  
TENPROC routine (phase 65)  
    description 187  
    diagram 585  
    flowchart 299,313  
TER.COD data-name 522  
TERM option  
    bit in COMMON 334  
    description 26  
    error and progress messages 44  
    phase 02 parameters 47  
    phase 6 introduction 141  
terminal error conditions 45  
TERMINATE statement 524,527  
TERMINATE verb, phase 1B processing of 6'  
termination, abnormal 31.32

TEST LINKAGE AREA field (PGT) 511  
 TEST option  
   BCDPN table for 466  
   DATATAB field 458  
   data set for 37-43  
   description 27  
   INIT3 processing for 513  
   interphase processing for 31  
   parameters for 47  
   phase 10 processing 56  
   phase 25 processing 85  
   phase 62 output for 162  
   phase 65 processing for 187-189  
 TESTSUBS routine  
   description 86  
   flowchart 250  
 text (see Data A-text; Data IC-text; Debug  
 text; DEF-text; dictionary entries;  
 E-text; Listing A-text; Optimization  
 A-text; Procedure A-text; Procedure  
 IC-text; REF-text; XREF-text)  
   definition 546  
   formats 399-444  
 TEXT filename  
   CMS FILEDEF for 532  
   CMS processing of 533  
 TGT (see Task Global Table)  
 TGT standard area reference element, phase  
   64 action 184  
 TGT storage allocation, phase 62 162-164  
 TGT-VN TABLE field (TGT)  
   description 507  
   location 505  
 TGTADTBL table  
   creation of 162  
   format 389-390  
   phase 6 151  
   use in phase 65 189  
 TGTINT routine  
   description 163  
   diagram of 581  
 THRTPROC routine  
   description 187  
   diagram 585  
 TIB (see Table Information Blocks)  
 TIB cell (COMMON) 325  
 TIB usage 315  
 Time-Sharing Option (TSO) 19  
 TOTTL table  
   Data Division processing 59  
   format 390  
   phase 10 processing 59  
   usage 313,314  
 TMCNTBSZ cell (COMMON) 331  
 TRANSFORM verb, phase 50 processing 132  
 Transient Area field (object module)  
   description 514  
   definition 546  
 translating LD entries 72  
 TRUNC option  
   description 27-28  
   switch for 333  
 TSMAX cell (COMMON)  
   description 326  
   use in phase 50 127  
   use in phase 6 143  
   use in phase 62 163  
 TWTWRO routine  
   description 88  
   Glossary building 88,597  
 TS2MAX cell (COMMON)  
   description 326  
   use in phase 6 143  
   use in phase 62 163  
 TS3MAX cell (COMMON)  
   description 328  
   use in phase 6 143  
   use in phase 62 163  
 TS4MAX cell (COMMON)  
   description 328  
   use in phase 6 143  
   use in phase 62 163  
 TWENPROC routine (phase 65)  
   description 202  
   diagram 585  
   flowchart 299  
 TXPNCH routine (phase 65)  
   description 187  
   diagram 585  
 TXTOUT table  
   format 390-391  
   usage 313,314  
   use in phase 45 118  
 TYPE clause, input to PROC01 routine 64  
 UNSTRING routine  
   description 118  
   flowchart 263  
 UNSTRING verb  
   compiler characteristics for 19  
   phase 4 processing for 104  
   phase 45 processing 118,119  
   phase 51 processing 137  
 UNTIL clause 137  
 UPDATE routine, diagram of 583  
 UPSIBIT switch (COMMON) 335  
 UPSTBL table  
   format 391  
   use in phase 22 75  
 USAGE clause, input to PROC01 routine 64  
 USE AFTER (BEFORE) STANDARD LABEL  
   declarative 327  
 USE AFTER STANDARD ERROR declarative 327  
 USE FOR DEBUGGING declarative 96,70  
 USE FOR DEBUGGING verbs 99-100  
 USE sentence  
   cell in COMMON for 327  
   debug processing 135  
   Declaratives processing 70  
   phase 1B processing of 61  
 USETBL table 391  
 USM-ROUT routine  
   generation of 64  
   logic of report writer subprogram 517  
 USNGTBL table 392  
 utility data sets 21  
 VALGEN routine 72  
 VALGRP table  
   format 392  
   input to phase 20 71  
   input to phase 22 75  
   output of phase 20 71

VALTRU table  
 description 95  
 format 392-393  
 input to phase 20 71  
 input to phase 22 75  
 output of phase 20 71

VALUE clauses, A-text generation for 77

variable-length field, definition 546

variable-length record (see Q-routines)

variable procedure-name (see VN)

variable procedure-name definition, optimization A-text format 440

VARLTBL table  
 format 393  
 input to phase 25 85  
 output of phase 22 75

VARYING clause 109

VARYTB table 393

VBREF option 29

VBSUM option 29

VCONDISP cell (COMMON) 332

verb analyzer routines  
 phase 4 111-116  
 phase 50 119-132  
 phase 51 135

verb information, Procedure IC-text format 421

VERB option 28

verb string, definition 546

verb strings 104

verbs, Procedure IC-text  
 P0 format 412  
 P1 format 421  
 P2 format 425

VIOVIRN cell (COMMON) 331

VIRCTR cell (COMMON)  
 description 325  
 initialization 131  
 use in phase 50 131  
 use in phase 6 149

VIRPTR table  
 format 394  
 optimizing virtuals 147-148  
 processing Procedure A-text elements 153  
 use in phase 62 166

VIRRTN routine (phase 6), optimizing virtuals 147

virtual, definition 546

virtual definition elements  
 description 144,146-148  
 Optimization A-text format 439

virtual EBCDIC names allocation 149,167

VIRTUAL EBCDIC NAMES field (PGT)  
 description 511  
 location 510

VIRTUAL field (PGT) 511

virtual identifying number 131

Virtual Machine Facility/370 529-541

virtual references  
 description 80  
 phase 21 processing 80  
 phase 64 action 184  
 Procedure A-text format 435

virtual storage 529

virtuals, optimization of 166

VLC field (TGT)  
 description 509  
 location 505  
 phase 6 processing 143  
 phase 62 counter for 163

VLCCTR cell (COMMON)  
 description 326  
 use in phase 6 143  
 use in phase 62 163

VM/370 529-541

VN (variable procedure-name)  
 phase 50 119  
 phase 51 134  
 phase 6 144  
 Procedure A-text format 434  
 Procedure IC-text format 429

VN definition element 183

VN field (TGT)  
 description 509  
 location 505  
 phase 62 counter for 164

VN priority table, phase 62 processing 165

VN reference element, phase 64 action 184

VNCTR cell (COMMON)  
 description 329  
 use in phase 62 168

VNI field (PGT)  
 allocation (phase 62) 168  
 description 511  
 location 510

VNILOC cell (COMMON)  
 description 329  
 use in phase 6 150

VNLOC cell (COMMON)  
 description 329  
 use in phase 6 143  
 use in phase 62 164

VNPNTBL table  
 format 394  
 phase 62 processing 165

VNPTY table  
 description 165  
 format 395

VNTBL table  
 description 104-109  
 format 395

VRBDN table  
 description 100  
 format 395-396

VRDEFTBL table 396

VSAM  
 debug FIB 467-469  
 FIB 80  
 interface 136,140  
 object code generation 140

VTINITVN cell (COMMON) 331

V2BUGSW cell (COMMON) 338

WCMAX cell (COMMON) 326

WG01 routine 33

work registers 155,156

WORKING CELLS field (TGT)  
 description 507  
 location 505

Working-Storage address elements  
 Data A-text format 409  
 description 160  
 phase 21 processing 80

phase 22 processing 77  
 phase 64 action 182  
 Working-storage dictionary entries, phase  
   22 processing 77  
 Working-Storage Section, phase 10  
   processing 60  
 WOUT routine  
   description 33  
   flowchart 201,202  
 WPCH routine, flowchart 203  
 WRITE routine (phase 00)  
   description 33  
   flowchart 201,202  
 WRITE routine (phase 63), diagram of 583  
 WRITE verb, debug processing 135  
 WRITEA routine  
   description 33  
   flowchart 201,202  
 WRITE5 routine (phase 21) 86  
 WRT-ROUT routine, generation of 64  
 WSDEF cell (COMMON) 327  
 WSECT routine flowchart 239  
 WSTSCT routine, flowchart 235  
  
 XAVAL table  
   description 127  
   format 396  
 XCTL macro instruction 46,47  
 XFCDNO cell  
   description 152-154  
   use in phase 64 185  
 XFTPT switch 125  
 XGDFFP switch 125  
 XGNCON switch 125  
 XGNSW switch 125  
 XGOVFL switch 125  
 XGSWT switch 125  
 XG2IRX switch 125  
 XINREG switch 125  
 XINTR table  
   description 124,125  
   format 396-397  
 XIS31 routine 120  
 XLITER switch 125  
 XLITZR switch 125  
 XOPMOD switch, format 125  
 XREF option  
   description 27  
   interphase processing for 31  
   output of phase 6 141,142  
   phase 02 parameters 47  
   phase 6A processing 190,191  
   phase 62 output 161  
   phase 64 processing for 179  
   switch for 333  
  
 XREF-text  
   formats 443  
   input/output operations 37-43  
   phases producing 316  
 XSA field (TGT)  
   description 509  
   location 505  
   phase 6 143  
   phase 62 counter for 164  
 XSACTR cell (COMMON)  
   description 329  
   use in phase 6 143  
   use in phase 62 164  
 XSASW field (TGT)  
   description 509  
   location 505  
   phase 6 143  
   phase 62 counter for 164  
 XSCOMP routine 121,122  
 XSCRPT table  
   description 120-122  
   format 397-398  
 XSPRO routine  
   description 132  
   flowchart 270  
 XSSNT table  
   description 122  
   format 398  
 XSUDB3 routine 122  
 XSWCTR cell (COMMON)  
   description 329  
   use in phase 51 137  
   use in phase 6 143  
   use in phase 62 164  
 XTEN routine 74  
 XY code 31  
 XZSWT switch, format 125  
  
 YBGN table, phase 05 52  
 Y1IRX+1 switch 125  
  
 ZWB option  
   description 27  
   phase 02 processing 47  
 ZZ code, linkage to phase 00 34  
  
 1ST-ROUT routine  
   generation of 64  
   logic of report writer subprogram 516  
   phase 1B processing 61



**International Business Machines Corporation**  
**Data Processing Division**  
**1133 Westchester Avenue, White Plains, New York 10604**  
**(U.S.A. only)**

**IBM World Trade Corporation**  
**821 United Nations Plaza, New York, New York 10017**  
**(International)**

IBM OS/VS COBOL Compiler  
Program Logic  
LY28-6486-2

**Reader's  
Comment  
Form**

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

**Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.**

If you would like a reply, please provide your name and address (including ZIP code).

**Fold on two lines, staple, and mail.** No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

Fold and Staple

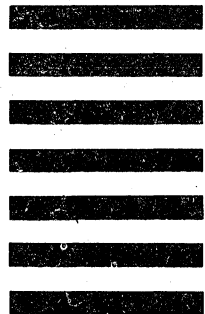
First Class Permit  
Number 6090  
San Jose, California

**Business Reply Mail**

No postage necessary if mailed in the U.S.A.

Postage will be paid by:

**IBM Corporation  
P.O. Box 50020  
Programming Publishing  
San Jose, California 95150**



Fold and Staple

OS/VS COBOL Compiler PLM Printed in U. S. A. LY28-6486-2



**International Business Machines Corporation  
Data Processing Division  
1133 Westchester Avenue, White Plains, New York 10604  
(U.S.A. only)**

**IBM World Trade Corporation  
821 United Nations Plaza, New York, New York 10017  
(International)**