

Licensed Material - Property of IBM

LY28-6423-1

**Program Product**

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

**Program Number: 5746-CB1**

**IBM**

Second Edition (September 1975)

This edition is a reprint of LY28-6423-0 incorporating changes released in Technical Newsletter LN28-1060 (dated March 15, 1974).

Changes are periodically made to the specifications herein; before using this publication in connection with the operation of IBM Systems, refer to the latest IBM System/360 and System/370 Bibliography, GA-6822, for editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Department J57, 1501 California Avenue, Palo Alto, California 94304. Comments become the property of IBM.

*Date of Publication:* March 22, 1974

*Form of Publication:* TNL LN28-1060 to LY28-6423-0

**Lister Option**

*New:* Programming

Produces a reformatted source listing with embedded cross-references. A reformatted source deck can also be obtained.

**VERBREF and VERBSUM Options**

*New:* Programming

Produces a listing of all verbs in a source program together with the number of times each appears. The statement numbers in which each verb appears can also be listed.

**COUNT Option**

*New:* Programming

Produces a listing containing the number of times each verb was executed, the procedure in which it appears, and the number of the statement in which it appears.

**MERGE Verb**

*New:* Programming

Combines from 2 through 8 identically sequenced files into one file in ascending or descending order according to keys contained in each record.

**Miscellaneous Changes**

*Maintenance:* Programming and Documentation

Minor changes have been made to the compiler and to the generated code. These are indicated in this publication by the bar to the left of the text.

---

Editorial changes that have no technical significance are not noted here.

Specific changes to the text made as of this publishing date are indicated by a vertical bar to the left of the text. These bars will be deleted at any subsequent republication of the page affected.



PREFACE

This publication describes the internal design of the IBM DOS/VS COBOL 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 parts:

- 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 chapter 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, the tables used by each phase, and a linkage editor map.
  - 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 showing the generated coding for input/output verbs.
  - A Glossary of special terms.
  - Foldout diagrams
  - An index.
- Effective use of this manual requires an extensive knowledge of the IBM Assembler Language, DOS/VS System Control, and the IBM DOS/VS COBOL language. Prerequisite and related publications include:
- IBM DOS/VS Operating Procedures, Order No. GC33-5378
- IBM OS/VS and DOS/VS Assembler Language Guide, Order No. GC33-4010
- IBM DOS/VS System Control Statements Reference, Order No. GC33-5376.
- IBM DOS/VS System Utilities Reference, Order No. GC33-5381.
- IBM DOS/VS Supervisor and I/O Macros Reference, Order No. GC33-5373.
- IBM DOS/VS Access Method Services, Order No. GC33-5382.
- IBM DOS/VS Data Management Guide, Order No. GC33-5372.
- Prerequisite Program Product documents include:
- IBM DOS Full American National Standard COBOL, Order No. GC28-6394.
- IBM DOS/VS COBOL Compiler and Library Programmer's Guide, Order No. SC28-6478.
- IBM DOS/VS COBOL Compiler and Library Installation Reference Material, Order No. SC28-6479.
- IBM DOS/VS COBOL Subroutine Library Program Logic, Order No. LY28-6424.

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."

CONTENTS

SECTION 1. INTRODUCTION . . . . .	13	GETDLM Routine . . . . .	39
Relationship To The DOS/VS System . . . . .	13	Identification Division . . . . .	39
Physical Characteristics . . . . .	13	Environment Division . . . . .	40
Operational Considerations . . . . .	14	Configuration Section . . . . .	40
Design of the Compiler . . . . .	14	OBJECT-COMPUTER Paragraph . . . . .	40
GROUP 1: Control and Initialization . . . . .	18	SPECIAL-NAMES Paragraph . . . . .	40
GROUP 2: Reformatting the Source Deck . . . . .	18	Input-Output Section . . . . .	40
GROUP 3: Translating the Identification, Environment, and Data Divisions . . . . .	18	File-Control Paragraph . . . . .	40
GROUP 4: Translating the Procedure Division . . . . .	19	VSAM File Processing . . . . .	40
GROUP 5: Assembly . . . . .	19	I-O-Control Paragraph . . . . .	41
GROUP 6: Final and Diagnostic Output . . . . .	20	Data Division . . . . .	41
Compiler Options . . . . .	20	File Section . . . . .	42
SECTION 2. METHOD OF OPERATION . . . . .	24	File Description Entries . . . . .	42
Phase 00 . . . . .	24	Sort Description Entries . . . . .	42
Receiving Control from and Returning It to the System . . . . .	24	Record Description Entries . . . . .	42
Processing Between Phases . . . . .	24	Working-Storage and Linkage Sections . . . . .	43
Phase Input/Output Requests . . . . .	28	Syntax Analysis . . . . .	43
Table and Dictionary Handling . . . . .	34	PHASE 12 . . . . .	44
Communications Area (COMMON) . . . . .	34	Operations in Other Phases . . . . .	44
Unrecoverable Error Conditions . . . . .	34	REPORT Clause . . . . .	44
Segmentation Operations . . . . .	34	Report Section Header . . . . .	44
PHASE 01 . . . . .	35	USE Sentences . . . . .	44
Compilation Parameters . . . . .	35	Procedure Division Verbs . . . . .	44
Buffer Size Determination . . . . .	35	Control-Field Save-Area Names . . . . .	45
Opening Files . . . . .	37	REDEFINES Clause . . . . .	45
Information Returned to Phase 00 . . . . .	37	Producing the report Writer Subprogram (RWS) . . . . .	45
Error Conditions . . . . .	37	Routine RDSCAN . . . . .	45
LIB Option Processing . . . . .	37	Routine PROC01 . . . . .	45
Federal Information Processing Standard (FIPS) Flagging . . . . .	38	Routine PROC02 . . . . .	47
PHASE 05 . . . . .	38	Routine FLUSH . . . . .	47
Input . . . . .	38	Routine GNSPRT . . . . .	47
Output . . . . .	38	Generating Error Messages . . . . .	47
Error Conditions . . . . .	38	Generating the Source Listing . . . . .	47
PHASE 06 . . . . .	38	Information for Later Phases . . . . .	47
Input . . . . .	38	PHASE 11 . . . . .	48
Output . . . . .	38	Encoding the Procedure Division . . . . .	48
PHASE 07 . . . . .	38	Processing Procedure-names . . . . .	48
Input . . . . .	38	Priority Checking for Segmentation . . . . .	48
Output . . . . .	38	Processing Verbs . . . . .	49
PHASE 08 . . . . .	38	Procedure Branching Verbs . . . . .	49
Input . . . . .	38	Input/Output Verbs . . . . .	49
Output . . . . .	38	Other Verbs . . . . .	49
Processing . . . . .	38	Processing Declaratives . . . . .	50
Error Conditions . . . . .	38	Entering Procedure-names in the Dictionary . . . . .	50
PHASE 10 . . . . .	39	Dummy Entry For Phase 80 . . . . .	50
Major Working Routines . . . . .	39	PHASE 20 . . . . .	52
GETCRD Routine . . . . .	39	Translating LD Entries into AIF-Text . . . . .	52
GETWD Routine . . . . .	39	Processing Elementary Items . . . . .	53
GETDLM Routine . . . . .	39	Processing Group Items . . . . .	53
Identification Division . . . . .	39	Producing Incomplete Data A-text . . . . .	53
Environment Division . . . . .	40	Processing File Section Entries . . . . .	53
Configuration Section . . . . .	40	Processing Errors . . . . .	53
OBJECT-COMPUTER Paragraph . . . . .	40	PHASE 22 . . . . .	55
SPECIAL-NAMES Paragraph . . . . .	40	Building Dictionary Entries . . . . .	55
Input-Output Section . . . . .	40	Dictionary Preprocessing . . . . .	55
File-Control Paragraph . . . . .	40	Completing Dictionary Entries . . . . .	57
VSAM File Processing . . . . .	40		
I-O-Control Paragraph . . . . .	41		
Data Division . . . . .	41		
File Section . . . . .	42		
File Description Entries . . . . .	42		
Sort Description Entries . . . . .	42		
Record Description Entries . . . . .	42		
Working-Storage and Linkage Sections . . . . .	43		
Syntax Analysis . . . . .	43		

Generating Data A-text . . . . .	58	ISTRUV Routine . . . . .	92
Q-Routine Generation . . . . .	58	IDERK Routine . . . . .	92
Processing Errors . . . . .	59	IDLH03 Routine . . . . .	92
Building Tables for Later Phases . . . . .	59	SORT, MERGE Routines . . . . .	92
		EOF Routine . . . . .	92
PHASE 21 . . . . .	60	GETNXT (GET13 and GET14) Routine . . . . .	92
Completing Dictionary Entries . . . . .	60	EXIT5 (EXIT PROGRAM) Routine . . . . .	92
Generating Required DTF's . . . . .	60	Branches to USE FOR DEBUGGING . . . . .	92
Clause Compatibility . . . . .	60	GENNOD Routine . . . . .	92
Common Parameters . . . . .	60	GENPAR Routine . . . . .	92
Record Size . . . . .	60	GENTIM Routine . . . . .	92
Record Form . . . . .	60	WRSYS4 Routine . . . . .	92
Selecting the DTF Generator . . . . .	61		
Determining the Number of DTF's . . . . .	62	PHASE 50 . . . . .	92
Pre-DTF Area . . . . .	62	Program Breaks . . . . .	92
DTFMT and DTFSD Pre-DTFs . . . . .	62	Verb Strings . . . . .	92
DTFDA, Random Access, Pre-DTF . . . . .	62	Verb Processing . . . . .	93
DTFDA, Sequential Access, Pre-DTF . . . . .	62	Resolving Subscripted and Indexed	
DTFIS Pre-DTF . . . . .	65	References . . . . .	93
DTFDU Pre-DTF . . . . .	65	Calculating Subscripted Addresses . . . . .	93
Pre-DTF Switch . . . . .	66	Literal Subscripts . . . . .	94
COBOL Indicators in DTF's . . . . .	66	Data-name Subscripts . . . . .	94
REWIND and COBOLRWD . . . . .	66	Mixed Literal and Data-name	
COBOL Bits . . . . .	66	Subscripts . . . . .	95
Writing Data A-text and P1-text . . . . .	67	Using and Optimizing Subscript	
DTFs For Associated Files . . . . .	67	References . . . . .	95
File Information Block (FIB) . . . . .	68	Indexed References . . . . .	96
Buffer Generation . . . . .	68	Arithmetic Verb Strings . . . . .	96
		Work Area . . . . .	97
PHASE 25 . . . . .	69	Compile-Time Arithmetic . . . . .	97
Phase 25 Processing for the Debug		Mode of Operation . . . . .	97
File . . . . .	69	Register and Storage Allocation . . . . .	98
Building the OBODOTAB Table . . . . .	69	Generating SRP Machine Instructions . . . . .	99
Building the DATATAB Table . . . . .	69	Generating A-Text . . . . .	99
		Literals and Virtuals . . . . .	99
PHASE 30 . . . . .	71	Handling Phase 51 Verb Strings . . . . .	103
Phase 30 Method of Operations . . . . .	71	Additional Processing for the Optimizer	
Glossary Building . . . . .	71	Option (OPT) . . . . .	103
Translation from P0-Text to P1-Text:			
PHCTRL Routine . . . . .	72	PHASE 51 . . . . .	104
READ Verb Strings: READFN Routine . . . . .	72	E-Text . . . . .	104
MERGE/SORT Verb Strings: SMERGE		Segmentation Control Breaks . . . . .	104
Routine . . . . .	73	PN, GN, and VN Definitions . . . . .	105
Statements with CORRESPONDING		Building PN and GN EQUATE Strings . . . . .	105
Options: CORR'TN Routine . . . . .	73	Building the PNUtbl Table . . . . .	105
SEARCH Verb Strings: STSRCH Routine . . . . .	74	Verb Strings . . . . .	105
Determining the Uniqueness of a		Input/Output Verbs . . . . .	106
Name: SEARCH Routine . . . . .	75	Other Nonarithmetic Verb Strings . . . . .	107
Replacing Names with Dictionary		Special Considerations for	
Attributes: GENOP Routine . . . . .	78	Nonarithmetic Verbs . . . . .	108
Error Processing: ERROR Routine . . . . .	78	Verbs Requiring Calls to	
		Object-Time Subroutines . . . . .	108
PHASE 40 . . . . .	81	DISPLAY Literals . . . . .	109
Translation Of P1-Text to P2-Text . . . . .	81	Generating System/370 Instructions . . . . .	109
Procedure-Names . . . . .	81	Generating Object Code To Process VSAM	
Verb Strings . . . . .	81	Files . . . . .	110
Examples . . . . .	81	Generating Calls to the ILBDVOC0	
MOVE Statement -- Subscripting . . . . .	81	and ILBDV100 COBOL Library	
DEBUG Card . . . . .	82	Subroutines . . . . .	110
ALTER Statement . . . . .	82		
PERFORM Statement . . . . .	84	PHASE 60 . . . . .	111
COMPUTE Statement . . . . .	87	Output of Phase 60 . . . . .	111
IF Statement . . . . .	88	Task Global Table Storage Allocation . . . . .	112
SEARCH ALL Statement . . . . .	89	Optimizing Storage for the Program	
Syntax Analysis and Verb Checking . . . . .	91	Global Table . . . . .	112
Method of Defining Verb Blocks . . . . .	92	Virtual References Definitions:	
Phase 40 Initialization Routine . . . . .	92	FILtbl Table . . . . .	114
ID5 Routine . . . . .	92	Building the VN Priority Table . . . . .	114



Optimizing PNs and GNs . . . . .	.114	Processing for Incremented Address	
Optimizing Literals and DISPLAY		(A4) Elements . . . . .	.139
Literals . . . . .	.115	Counters Used in Phase 63 . . . . .	.139
Optimizing Virtuals . . . . .	.116	Building the QGNTBL Table . . . . .	.140
Allocating Storage for the Program		Making Entries in the RLDTBL Table . . . . .	.140
Global Table . . . . .	.116	Processing in a Segmented Program . . . . .	.140
Procedure A-Text Processing . . . . .	.118	Processing at End of File . . . . .	.140
Processing in a Segmented Program	.121	PHASE 64 . . . . .	.141
Execution Time Base Register		Output of Phase 64 . . . . .	.141
Assignment . . . . .	.122	Completing the RLDTBL Table . . . . .	.141
Processing Data A-Text, E-Text, and		Completing The Machine Language Program	141
DEF-Text . . . . .	.123	Initialization Routines . . . . .	.143
Processing the RLDTBL Table . . . . .	.123	RLDTBL Table Processing . . . . .	.143
Initialization Routines . . . . .	.123	PHASE 61 . . . . .	.144
PHASE 65 . . . . .	.126	Producing a Source Ordered	
Processing the Flow Option . . . . .	.126	Cross-Reference Listing . . . . .	.144
Common Processing for the STATE and the		Producing an Alphabetically Ordered	
SYMDMP Options . . . . .	.126	Cross-Reference Listing . . . . .	.145
Processing Debug-text . . . . .	.126	PHASE 70 . . . . .	.146
Building the PROCTAB Table . . . . .	.126	Input from Prior Phases . . . . .	.146
Building the SEGINDX Table . . . . .	.127	Phase 70 Error Processing . . . . .	.146
Further Processing for the STATE Option	127	The PARTBL and EACTBL Tables . . . . .	.146
Further Processing for the SYMDMP		The PHxERR Table . . . . .	.146
Option . . . . .	.127	Generating Messages . . . . .	.147
Building the CARDINDX Table . . . . .	.127	Error Message Listing . . . . .	.147
Building the PROCINDX Table . . . . .	.127	PHASE 80 . . . . .	.148
Debug File Processing . . . . .	.127	Input . . . . .	.148
Final Processing . . . . .	.128	Output . . . . .	.148
PHASE 62 . . . . .	.129	Processing . . . . .	.148
Output of Phase 62 . . . . .	.129	Scanning The Source Program . . . . .	.148
Allocating Storage for the Task Global		Generating Diagnostic Messages . . . . .	.148
Table (TGT) . . . . .	.130	Writing the Source Program . . . . .	.148
Optimizing and Allocating Storage for		SECTION 3. PROGRAM ORGANIZATION . . . . .	.149
the Program Global Table (PGT) . . . . .	.130	Flowcharts . . . . .	.149
Optimizing and Building Tables . . . . .	.130	Explanation of Flowchart Symbols . . . . .	.150
DTF Virtuals and VN Priority Table	.130	SECTION 4. DIRECTORY . . . . .	.259
Optimizing Literals and DISPLAY		Flowchart Label Directory . . . . .	.259
Literals . . . . .	.130	Tables Used By Phases . . . . .	.263
Optimizing Virtuals . . . . .	.130	Linkage Editor Map . . . . .	.265
Processing for PNs and GNs . . . . .	.130	SECTION 5. DATA AREAS . . . . .	.275
Allocating Storage for the PGT . . . . .	.131	Communication Region . . . . .	.275
Optimizing Register Assignments . . . . .	.132	TABLE FORMATS . . . . .	.287
Permanent Register Assignments . . . . .	.132	TEXT FORMATS . . . . .	.341
Temporary Register Assignments . . . . .	.132	Data IC-Text . . . . .	.341
Optimizing and Allocating Storage for		ATF-text . . . . .	.348
the Procedure Division . . . . .	.133	Data A-text . . . . .	.349
Building the PNLABTBL and GNLABTBL		Procedure IC-text (P0 Format) . . . . .	.351
Tables . . . . .	.134	Procedure IC-text (P1 Format) . . . . .	.359
Incrementing the ACCUMCTR Counter	.134	Procedure IC-Text (P2 Format) . . . . .	.364
PHASE 63 . . . . .	.138	Procedure A-Text . . . . .	.373
Initialization of Phase 63 . . . . .	.138	Optimization A-Text . . . . .	.378
Constructing Procedure A1-Text . . . . .	.138	Procedure A1-Text . . . . .	.380
Control Routine . . . . .	.138	Listing A-Text . . . . .	.381
Processing Programs with One		E-Text . . . . .	.382
Procedure Block . . . . .	.138	XREF-Text . . . . .	.383
Processing for Branch Instructions . . . . .	.138	Debug-Text . . . . .	.384
Processing for Optimization		DICTIONARY ENTRY FORMATS . . . . .	.385
Information Elements (C001-C007) . . . . .	.139	Procedure-name (Paragraph) Entry . . . . .	.385
Processing for RPT-ORIGIN (D4)			
Element . . . . .	.139		
Processing for Address Reference (78)			
Elements . . . . .	.139		
Processing for Address Increment (80)			
Elements . . . . .	.139		

Procedure-name (Section) Entry . . . . .	385	TAMEIN Routine . . . . .	426
FD Entry . . . . .	386	PRIME Routine . . . . .	427
FD ENTRY FOR VSAM FILES . . . . .	387	TBGETSPC Routine . . . . .	427
SD Entry . . . . .	387	MOVDIC Routine . . . . .	427
RD Entry . . . . .	388	DICSPC Routine . . . . .	427
LD Entry . . . . .	388	STATIC Routine . . . . .	427
Condition-name Entry . . . . .	388	TABREL Routine . . . . .	428
Index-name Entry . . . . .	389	INSERT Routine . . . . .	428
DEBUG FILE TABLES . . . . .	395	TAMEOP Routine . . . . .	428
PROGSUM Table . . . . .	396	TBSPILL Routine . . . . .	428
ORODOTAE Table . . . . .	397	TBWRITE Routine . . . . .	428
DATATAB Table . . . . .	398	TBREADIC Routine . . . . .	429
PROCTAB Table . . . . .	404	GETALL Routine . . . . .	429
CARDINDX Table . . . . .	404	APPENDIX B: OBJECT MODULE . . . . .	430
SEGINDX Table . . . . .	405	Initialization 1 Routine (INIT1) . . . . .	430
PROCINDX Table . . . . .	405	Working-Storage . . . . .	430
VSAM File Information Block (FIB) . . . . .	405	DTF's and Buffers . . . . .	430
SECTION 6. DIAGNOSTIC AIDS . . . . .	409	Task Global Table (TGT) . . . . .	430
Error Message Listing . . . . .	409	Program Global Table (PGT) . . . . .	435
Dump Produced For Disaster Level Error Messages . . . . .	409	Report Writer . . . . .	436
Abnormal Termination During Compilation . . . . .	409	Procedure Division . . . . .	436
Location of Information in Storage . . . . .	409	Q-Routines . . . . .	436
Current Phase . . . . .	409	COUNT Table . . . . .	437
Current Record . . . . .	410	Initialization 2 Routine (INIT2) . . . . .	437
Tables Used by Phases . . . . .	410	Initialization 3 Routine (INIT3) . . . . .	437
Compiler Buffers and Their Contents . . . . .	410	FLOW TRACE Table . . . . .	438
Register Usage . . . . .	411	PROCTAB Table (PROCTAB) . . . . .	438
Register Saving . . . . .	418	SEGINDX Table (SEGINDX) . . . . .	438
Input/Output Error Messages . . . . .	418	Transient Area (Segmented Program) . . . . .	438
Erroneous Compiler Output . . . . .	418	APPENDIX C: REPORT WRITER SUBPROGRAM . . . . .	440
Storage Layout . . . . .	418	Structure of the Report Writer Subprogram (RWS) . . . . .	440
Locating a DTF . . . . .	419	Elements of a Report Writer Subprogram (RWS) . . . . .	440
Locating Data . . . . .	419	Fixed Routines . . . . .	440
Register Usage . . . . .	419	1ST-ROUT Routine . . . . .	440
Error Messages . . . . .	420	LST-ROUT Routine . . . . .	440
Linkage Editor Phase Map . . . . .	420	WRT-ROUT Routine . . . . .	440
Diagnostic Assistance . . . . .	420	Parametric Routines . . . . .	440
APPENDIX A: LABEL TABLE DIRECTORY . . . . .	421	USM-ROUT Routine . . . . .	441
ACCESS Dictionary Handling Routines . . . . .	421	CTB-ROUT Routine . . . . .	441
Organization of the Dictionary . . . . .	421	ROL-ROUT Routine . . . . .	441
Area for the Dictionary . . . . .	422	RST-ROUT Routine . . . . .	441
Initialization of ACCESS Routines . . . . .	422	SAV-ROUT Routine . . . . .	441
ACCESS Routines . . . . .	422	RET-ROUT Routine . . . . .	441
ENTNAM (Enter Attributes Given Name) . . . . .	422	INT-ROUT Routine . . . . .	441
ENTPTR (Enter Attributes Given Pointer) . . . . .	423	ALS-ROUT Routine . . . . .	441
GETPTR (Get Pointer) . . . . .	423	RLS-ROUT Routine . . . . .	446
ENTDEL (Enter Delimiter Pointer) . . . . .	423	Group Routines . . . . .	446
LATRNM (Locate Attributes Given Name) . . . . .	423	RPH-ROUT Routine . . . . .	446
LATRPT (Locate Attributes Given Pointer) . . . . .	424	RPF-ROUT Routine . . . . .	446
LOCNXT (Locate Next Entry) . . . . .	424	CTH-ROUT Routine . . . . .	446
LDELNM (Locate Delimiter Given Name) . . . . .	424	CTF-ROUT Routine . . . . .	446
LATACP (Locate Attributes Using ACCESS Pointer) . . . . .	424	CHF-ROUT Routine . . . . .	446
LATGRP (Locate Attributes Given Group Pointer) . . . . .	425	CFP-ROUT Routine . . . . .	446
Table Handling with TAMER . . . . .	425	PGH-ROUT Routine . . . . .	446
Control Fields . . . . .	425	PGF-ROUT Routine . . . . .	446
TIB (Table Information Block) . . . . .	425	DET-ROUT Routine . . . . .	447
TAMM (Table Area Management Map) . . . . .	425	Data-names . . . . .	447
MASTAM (Master TAMM Table) . . . . .	426	COBOL Word Data-names . . . . .	447
How Space Is Assigned . . . . .	426	Nonstandard Data-names . . . . .	447
		Special Report Writer Verbs . . . . .	448
		Response to Procedure Division Verbs . . . . .	449
		Finding the Elements of a Report Writer Subprogram (RWS) . . . . .	449
		Locating Data Items in a Storage Dump . . . . .	449

Locating Data Items in the Object		CLOSE Coding . . . . .	.457
Module . . . . .	.449	CLOSE REEL Coding (For DTFSD, DTFMT,	
Locating Routines in a Storage Dump	.449	and DTFDA Sequential Input Only) . . .	.459
Locating Routines in the Object		READ Coding . . . . .	.460
Module . . . . .	.452	WRITE and REWRITE Coding . . . . .	.463
Locating DET-ROUT and USM-ROUT		SEEK Coding . . . . .	.467
Routines . . . . .	.452	START Coding . . . . .	.467
Locating CTF-ROUT and CTH-ROUT		DISPLAY Coding . . . . .	.468
Routines . . . . .	.453	ACCEPT Coding: . . . . .	.468
RWS Logic Flowcharts . . . . .	.453	USE Coding . . . . .	.469
APPENDIX D: GENERATED CODE FOR		GLOSSARY . . . . .	.470
INPUT/OUTPUT VERBS . . . . .	.454	DIAGRAMS . . . . .	.474
OPEN Coding . . . . .	.454	INDEX . . . . .	.509



FIGURES

Figure 1. Compiler Storage Layout . . .	13	Figure 37. Effect of a PERFORM Statement . . . . .	85
Figure 2. Compiler Output . . . . .	15	Figure 38. Execution of a PERFORM Statement . . . . .	86
Figure 3. List of Internal Compiler Texts (Part 1 of 3) . . . . .	16	Figure 39. Flow of Control for Statements in Figure 38 . . . . .	87
Figure 4. Contents of Work Files Used in Translating the Data Division . . . . .	20	Figure 40. Evaluation of a COMPUTE Statement . . . . .	88
Figure 5. Linkage Codes to Phase 00 . . . . .	25	Figure 41. Strings Resulting from a COMPUTE Statement . . . . .	88
Figure 6. Optional Phase Processing . . . . .	26	Figure 42. Evaluation of a Nested IF Statement . . . . .	89
Figure 7. Flow of Control at End of Compilation (Normal and Abnormal) . . . . .	27	Figure 43. Output for a SEARCH ALL Statement . . . . .	90
Figure 8. Flow of Control for Processing Between Phases . . . . .	28	Figure 44. Flow of Execution for a SEARCH ALL Statement . . . . .	91
Figure 9. Compiler File Handling (Part 1 of 6) . . . . .	29	Figure 45. Parameter Cells for the A-Text Generator (Part 1 of 4) . . . . .	100
Figure 10. Buffer Assignments . . . . .	36	Figure 46. Analysis of an ON Statement . . . . .	107
Figure 11. Phase 12 Input/Output Flow . . . . .	46	Figure 47. Analysis of a DISPLAY Verb . . . . .	109
Figure 12. Entering PNTABL and PNQTLB Information into the Dictionary . . . . .	51	Figure 48. Use of Counters in COMMON To Allocate Space in the TGT for Variable-Length Fields . . . . .	113
Figure 13. Phase 20 Input/Output Flow . . . . .	52	Figure 49. PNUTBL, PNTBL, and GNTBL Tables at the Beginning of Optimization Processing . . . . .	114
Figure 14. Phase 22 Input/Output Flow . . . . .	56	Figure 50. GNTBL Table After PN and GN Equate Strings Have Been Processed . . . . .	114
Figure 15. RECPORM Parameters Supported . . . . .	61	Figure 51. GNTBL Table After the Relative Numbers Have Been Assigned . . . . .	115
Figure 16. OPEN Options and DTFs . . . . .	62	Figure 52. CONTBL, CONDIS, and LTLTBL Tables After Processing Literals . . . . .	115
Figure 17. Pre-DTF for DTFMT and DTFSD . . . . .	63	Figure 53. CVIRTB and VIRPTR Tables After Processing Virtuals . . . . .	116
Figure 18. Pre-DTF for DTFDA, Random Access, Absolute Addressing . . . . .	63	Figure 54. VIRPTR Table After VIRTUAL Allocation . . . . .	117
Figure 19. Pre-DTF for DTFDA, Random Access, Relative Addressing . . . . .	64	Figure 55. PNTBL Values After PGT Allocation . . . . .	117
Figure 20. Pre-DTF for DTFDA, Sequential Access, Absolute Addressing . . . . .	64	Figure 56. Processing Procedure A-text Elements (Part 1 of 3) . . . . .	119
Figure 21. Pre-DTF for DTFDA, Sequential Access Relative Addressing . . . . .	65	Figure 57. Contents of SYS004 When Read by Phase 60 . . . . .	123
Figure 22. Pre-DTF for DTFIS . . . . .	65	Figure 58. Processing Data A-text, E-text, and DEF-text (Part 1 of 2) . . . . .	124
Figure 22A. Pre-DTF for DTFDU . . . . .	66	Figure 59. Optimizing Assignment of Registers 14 and 15 . . . . .	133
Figure 23. COBOLRWD and REWIND bits . . . . .	66	Figure 60. Processing for Optimization Information Elements (Part 1 of 3) . . . . .	135
Figure 24. COEOL bit settings . . . . .	67	Figure 61. Processing of Procedure A1-Text* . . . . .	142
Figure 25. DTF Chaining for an Associated File with Three Functions . . . . .	68	Figure 62. Tables Used by Phases (Part 1 of 2) . . . . .	263
Figure 26. P1-text Resulting from an ADD CORRESPONDING Option . . . . .	74	Figure 63. SYS005 (Debug File) . . . . .	395
Figure 27. P1-text Resulting from a MOVE CORRESPONDING Option . . . . .	74	Figure 64. POINT Table Entry Format . . . . .	410
Figure 28. P1-Text for SEARCH Format-1 (Part 1 of 2) . . . . .	75	Figure 65. Buffer Control Blocks for Buffers 1-6 . . . . .	411
Figure 29. P1-Text for SEARCH Format-2 . . . . .	77	Figure 66. Buffer Control Block Format . . . . .	411
Figure 30. P1-text Written for Condition-String Without VALUE...THRU Clause . . . . .	79	Figure 67. Compiler Register Usage . . . . .	411
Figure 31. P1-text Written for Condition-String with VALUE...THRU Clause . . . . .	80	Figure 68. Example of Storage Usage During Execution . . . . .	419
Figure 32. Tables and Output for a MOVE Statement . . . . .	82	Figure 69. Register Usage at Execution . . . . .	419
Figure 33. DBGIBL Entries and P2-text for DEBUG . . . . .	82		
Figure 34. Table Entries and Output for ALTER Statements . . . . .	83		
Figure 35. Execution of an ALTER Statement . . . . .	84		
Figure 36. Flow of Control for Statements in Figure 35 . . . . .	84		

Figure 70. Register Usage at Execution when OPT is Specified . . . . .	.420	Figure 77. PGT Fields . . . . .	.435
Figure 71. Example of a Phase Map . . . . .	.420	Figure 78. INIT2 Coding . . . . .	.436
Figure 72. Arrangement of Tables and Dictionary Sections in a Contiguous Region . . . . .	.421	Figure 79. INIT3 Coding . . . . .	.438
Figure 73. Storage Map of Object Module Fields . . . . .	.430	Figure 80. Logic of the Generated Report Writer Subprogram (Part 1 of 4) .	.442
Figure 74. INIT1 Coding . . . . .	.431	Figure 81. First GENERATE Statement Logic Flow . . . . .	.450
Figure 75. TGT Fields . . . . .	.432	Figure 82. Logic Flow of All GENERATE Statements After the First . . . . .	.451
Figure 76. Debug Table Formats . . . . .	.433	Figure 83. TERMINATE Statement Logic Flow . . . . .	.452
		Figure 84. RWS GN Numbers . . . . .	.452

DIAGRAMS

Diagram 1. Overview of the Compiler (Part 1 of 2) . . . . .	.475	Diagram 2. Part 7. Method of Operation: Optimization of Object Module (Optional) . . . . .	.491
Diagram 2. Part 1. Method of Operation: Table of Contents . . . . .	.479	Diagram 2. Part 8. Method of Operation: Debug Data Set Creation (Optional) . . . . .	.493
Diagram 2. Part 2. Method of Operation: Overview . . . . .	.481	Diagram 2. Part 9. Method of Operation: Error Messages, Diagnostics, and Cross-Reference Listings . . . . .	.495
Diagram 2. Part 3. Method of Operation: Control and Input/Output . .	.483	Diagram 3. Phase 25 Operations . . . . .	.497
Diagram 2. Part 4. Method of Operation: Identification, Environment, Data Division Translation	485	Diagram 4. Phase 25 Processing for the OBODOTAB Table . . . . .	.499
Diagram 2. Part 5. Method of Operation: Procedure Division Translation . . . . .	.487	Diagram 5. Phase 30 Operations . . . . .	.501
Diagram 2. Part 6. Method of Operation: Object Module Production . .	.489	Diagram 6. Phase 65 Operations . . . . .	.503
		Diagram 7. Phase 62 Operations . . . . .	.505
		Diagram 8. Phase 63 Operations . . . . .	.507

Chart AA. Phase 00 (ILACBL00):	Chart IA. Phase 50 (ILACBL50):
Overall Logic (Part 1 of 7) . . . . .151	Overall Logic . . . . .191
Chart AA. Phase 00: Overall Logic	Chart LB. Phase 50: GETNXT (Part 1
(Part 2 of 7) . . . . .152	of 2) . . . . .192
Chart AA. Phase 00: Overall Logic	Chart LB. Phase 50: GETNXT (Part 2
(Part 3 of 7) . . . . .153	of 2) . . . . .193
Chart AA. Phase 00: Overall Logic	Chart LC. Phase 50: A-text Generator .194
(Part 4 of 7) . . . . .154	Chart LD. Phase 50: XSPRO and KILSUB
Chart AA. Phase 00: Overall Logic	Routines . . . . .195
(Part 5 of 7) . . . . .155	Chart LE. Phase 50: DBGTEST . . . . .196
Chart AA. Phase 00: Overall Logic	Chart MA. Phase 51 (ILACBL51):
(Part 6 of 7) . . . . .156	Overall Logic . . . . .197
Chart AA. Phase 00: Overall Logic	Chart MB. Phase 51: DBGTEST . . . . .198
(Part 7 of 7) . . . . .157	Chart MC. Phase 51: GETNXT Routine . .199
Chart BA. Phase 01 (ILACBL01):	Chart MD. Phase 51: PUTDEF Routine . .200
Overall Logic (Part 1 of 2) . . . . .158	Chart ME. Phase 51: SET Verb
Chart BA. Phase 01 (ILACBL01): Overall	Analyzer, Format 1 . . . . .201
Logic (Part 2 of 2) COPY/BASIS	Chart MF. Phase 51: MOVE4 . . . . .202
Functions . . . . .159	Chart MG. Phase 51: SETLEN and
Chart CA. Phase 10 (ILACBL10):	LOADLT . . . . .203
Overall Logic . . . . .160	Chart MH. Phase 51: SET Verb
Chart DA. Phase 11 (ILACBL11):	Analyzer, Format 2 . . . . .204
Overall Logic . . . . .161	Chart MI. Phase 51: PERFORM and
Chart EA. Phase 12 (ILACBL12):	TRANSFORM . . . . .205
Overall Logic . . . . .162	Chart MJ. Phase 51: DISPLAY and
Chart EB. Phase 12: FLUSH Routine . .163	EQUATE . . . . .206
Chart FA. Phase 20 (ILACBL20):	Chart MK. Phase 51: IMINIT, IMGEN,
Overall Logic . . . . .164	RESET, and EXITPGM . . . . .207
Chart FB. Phase 20: FILEST Routine . .165	Chart ML. Phase 51: DEBUG, READ,
Chart FC. Phase 20: WST SCT, LINKST,	TRACE, and GOBACK Routines . . . . .208
and REPORT Routines . . . . .166	Chart MM. Phase 51: IF-Index Routines .209
Chart FD. Phase 20: LDTEXT Routine . .167	Chart MN. Phase 51: GO, and GODEPM . .210
Chart GA. Phase 21 (ILACBL21):	Chart MO. Phase 51: GODEPL and GO
Overall Logic . . . . .168	DEPENDING . . . . .211
Chart GB. Phase 21: RECORD CONTAINS	Chart MP. Phase 51: Nonnumeric IF
Clause Processing . . . . .169	(IFANAL) and Class Test (CLANAB)
Chart GC. Phase 21: BLOCK CONTAINS	Processors . . . . .212
Clause Processing . . . . .170	Chart MQ. Phase 51: SEGAL and SEGAL3 .213
Chart GD. Phase 21: RECORDING MODE	Chart MR. Phase 51: A-text
Clause Processing . . . . .171	Generator, and GATXTC and GATXTV
Chart GE. Phase 21: BUFGEN (Part 1	Routines . . . . .214
of 2) . . . . .172	Chart NA. Phase 60 (ILACBL60):
Chart GE. Phase 21: BUFGEN (Part 2	Overall Logic . . . . .215
of 2) . . . . .173	Chart NB. Phase 60: PH6 Routine . . .216
Chart HA. Phase 22 (ILACBL22):	Chart NC. Phase 60: PRFTWO Routine . .217
Overall Logic . . . . .174	Chart ND. Phase 60: SE6000 Routine . .218
Chart HB. Phase 22: FSECT Routine . .175	Chart NE. Phase 60: PDATEX (Part 1
Chart HC. Phase 22: WSECT and LSECT	of 2) . . . . .219
Routines . . . . .176	Chart NE. Phase 60: PDATEX (Part 2
Chart HD. Phase 22: RSECT Routine . .177	of 2) . . . . .220
Chart HE. Phase 22: LDTXT Routine . .178	Chart OA. Phase 62: Overall Logic . .221
Chart HF. Phase 22: READF4 Routine . .179	Chart OB. Phase 62: PH6 . . . . .222
Chart HG. Phase 22: DICTBD Routine . .180	Chart OC. Phase 62: PRFTWO . . . . .223
Chart IA. Phase 25: Overall Logic . .181	Chart OD. Phase 62: SE6000 (Part 1
Chart IB. Phase 25: ODOBLD,	of 2) . . . . .224
BLDOBODO, and ENPP1 . . . . .182	Chart OD. Phase 62: SE6000 (Part 2
Chart IC. Phase 25: BEGPASS . . . . .183	of 2) . . . . .225
Chart ID. Phase 25: TESTSUBS and	Chart PA. Phase 63 (ILACBL63):
SETNAMS . . . . .184	Overall Logic . . . . .226
Chart JA. Phase 30 (ILACBL30):	Chart PE. Phase 63: BRANCH . . . . .227
Overall Logic . . . . .185	Chart PC. Phase 63: GNDEF . . . . .228
Chart JB. Phase 30: GLOSRY Routine . .186	Chart PD. Phase 63: PNDEF . . . . .229
Chart JC. Phase 30: PHCTRL Routine . .187	Chart PE. Phase 63: ADREF and ADINCR .230
Chart KA. Phase 40 (ILACBL40):	Chart PF. Phase 63: C1REF, PNREF,
Overall Logic . . . . .188	and GNREF . . . . .231
Chart KB. Phase 40: IF Processing . .189	Chart QA. Phase 64 (ILACBL64):
Chart KC. Phase 40: PERFORM	Overall Logic . . . . .232
Processing . . . . .190	

Chart QB. Phase 64: ADREF, RC4, RC8C, and RD001 . . . . .	.233	Chart UH. WRT-ROUT Subroutine, Report Writer Subprogram . . . . .	.246
Chart RA. Phase 65 (ILACBL65): Overall Logic . . . . .	.234	Chart UI. RPH-ROUT Subroutine, Report Writer Subprogram . . . . .	.247
Chart RB. Phase 65: Debug-text Element Processors (TENPROC, IWENPROC, GTEQ10K) . . . . .	.235	Chart UJ. RST-ROUT Subroutine, Report Writer Subprogram . . . . .	.248
Chart SA. Phase 61 (ILACBL61): Overall Logic (Part 1 of 3) . . . . .	.236	Chart UK. ROL-ROUT Subroutine, Report Writer Subprogram . . . . .	.249
Chart SA. Phase 61 (ILACBL61): Overall Logic (Part 2 of 3) . . . . .	.237	Chart UL. ALS-ROUT Subroutine, Report Writer Subprogram . . . . .	.250
Chart SA. Phase 61 (ILACBL61): Overall Logic (Part 3 of 3) . . . . .	.238	Chart UM. PGH-ROUT Subroutine, Report Writer Subprogram . . . . .	.251
Chart TA. Phase 70 (ILACBL70): Overall Logic . . . . .	.239	Chart UN. SAV-ROUT Subroutine, Report Writer Subprogram . . . . .	.252
Chart UA. DET-ROUT Subroutine, Report Writer Subprogram . . . . .	.240	Chart UO. CTF-ROUT Subroutine, Report Writer Subprogram . . . . .	.253
Chart UB. 1ST-ROUT Subroutine, Report Writer Subprogram . . . . .	.241	Chart UP. RET-ROUT Subroutine, Report Writer Subprogram . . . . .	.254
Chart UC. CTB-ROUT Subroutine, Report Writer Subprogram . . . . .	.242	Chart UQ. INT-ROUT Subroutine, Report Writer Subprogram . . . . .	.255
Chart UD. USM-ROUT Subroutine, Report Writer Subprogram . . . . .	.243	Chart UR. LST-ROUT Subroutine, Report Writer Subprogram . . . . .	.256
Chart UF. PGF-ROUT Subroutine, Report Writer Subprogram . . . . .	.244	Chart US. CFF-ROUT Subroutine, Report Writer Subprogram . . . . .	.257
Chart UG. RLS-ROUT Subroutine, Report Writer Subprogram . . . . .	.245	Chart UT. RPF-ROUT Subroutine, Report Writer Subprogram . . . . .	.258



SECTION 1. INTRODUCTION

The IBM DOS/VS COBOL Compiler analyzes source modules written in the COBOL language and translates them into object modules. This publication describes the design and function of the compiler and the characteristics of the object program which it produces.

RELATIONSHIP TO THE DOS/VS SYSTEM

A COBOL compilation is a run unit under the control of the IBM DOS/VS System. The compiler uses the DOS/VS System Control Program for input/output and other services. For a general description of the System Control Program, refer to IBM DOS/VS System Management Guide, Order No. GC33-5371, and to IBM DOS/VS Supervisor and Input/Output Macros Reference, Order No. GC33-5373.

PHYSICAL CHARACTERISTICS

The compiler consists of 25 phases; from 10 to 14 of these phases perform the actual transformation of a source module into an object program. Phases 05, 06, 07, and 08 are called only if an LST card, which specifies the Lister option, is present. Phase 12 is called only if a Report Section appears in the Data Division. If optimization of the object code has been requested through the OPT option, phases 62, 63, and 64 replace phase 60. 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, 61 and 70 are also optional phases: phases 25 and 65 generate debugging information for the SYMDMP, FLOW, and/or STATE options; phase 61 produces a cross-reference listing if the user requests one; and phase 70 is 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 resolves BASIS and COPY statements and performs compiler initialization.

The phases are organized into an overlay structure; but the overlays are controlled

through phase 00. Phase 00 is resident in lower storage throughout compilation. It links to other phases as they are needed. The linkage sequence is as follows:

Phase 00 links to phases 01, 05 (if LST), 06 (if LIST), 07 (if LST), 08 (if LST), 10, 12, 11, 20, 21, 22, 25 (if SYMDMP), 30, 40, 50, 51, 60, 62, 63, 64, 65 (if SYMDMP), 61 (if XREF, SXREF, VERBSUM, or VERBREF), 70 (if any diagnostic messages are to be written), and 80 (if LVL).

Most phases occupy the portion of virtual storage contiguous to phase 00; phase 80 overlays phase 00 when LVL is in effect.

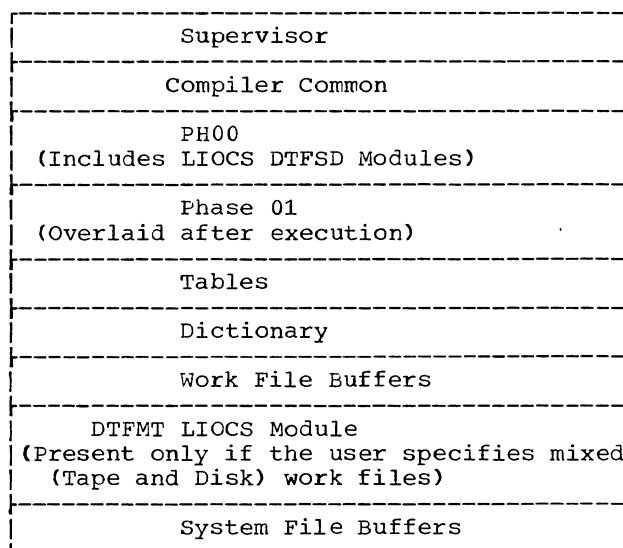


Figure 1. Compiler Storage Layout

Phase 00 initially occupies 12-13K bytes of storage (where K=1024 decimal). Phase 00 occupies 11K bytes when phases 40 through 65 are processing; it occupies 7K bytes when phase 70 is loaded. Phase 80 overlays phase 00. In addition, tables created by other phases require space in storage. The amount varies greatly with each compilation. The approximate sizes of the other phases are:

- Phase 01 - 18K bytes
- Phase 05 - 14K bytes
- Phase 06 - 4K bytes
- Phase 07 - 5K bytes
- Phase 08 - 12K bytes
- Phase 10 - 30K bytes
- Phase 11 - 28K bytes
- Phase 12 - 33K bytes

Phase 20 -	22K bytes
Phase 21 -	35K bytes
Phase 22 -	25K bytes
Phase 25 -	6K bytes
Phase 30 -	14K bytes
Phase 40 -	44K bytes
Phase 50 -	40K bytes
Phase 51 -	40K bytes
Phase 60 -	32K bytes
Phase 61 -	8K bytes
Phase 62 -	17K bytes
Phase 63 -	9K bytes
Phase 64 -	22K bytes
Phase 65 -	6K bytes
Phase 70 -	49K bytes
Phase 80 -	28K bytes

texts, each more similar to machine code than the last, until it produces an object module suitable for input to the linkage editor. Figure 3 shows the various texts and the phases which build them, as well as the files on which they are passed from phase to phase.

Each translation phase scans its input for errors and takes appropriate action. Unless a termination error occurs, it translates part or all of its input into a new text, further alters what had been begun by a previous phase, or sometimes passes along existing text to a later phase. This process continues through phase 60 or phase 64, when the object module is assembled.

#### OPERATIONAL CONSIDERATIONS

Input to the DOS/VS compiler consists of a source program written in the DOS/VS COBOL language. Input is read from either the SYSIPT or the SYSSLB file.

Output depends on the options that are in effect for the particular compilation. These options may be the installation default options set within the compiler or the overrides set by the programmer on the OPTION statement, the CBL card or the LST card. The "Compiler Options" section in this chapter describes each of these options. Figure 2 summarizes the output produced by the compiler. The phases are shown in the order in which they are loaded.

#### DESIGN OF THE COMPILER

The compiler translates the source program into a series of internal compiler

The phases communicate by placing data in a communications area (COMMON) in storage, entries in the dictionary and other tables, and text on work files SYS001, SYS002, SYS003, and SYS004. When the SYMDMP, VERBREF, or VERBSUM option is specified, an additional work file, SYS005, is also used. When LVL is specified, SYS006 is used. Some of the tables are built, used, and released within a single phase, and others are passed to later phases which may or may not immediately follow.

Diagram 1 shows the output and the flow of information and control from phase to phase; it is designed to supplement the rest of this discussion. The phases are discussed below in functional groups rather than in numerical order.

Phase	File	Contents	Options Required
01	SYSLST SYSLST SYSLST <sup>6</sup>	List of options Error messages resulting from compiler initialization Source coding	LIST LIST LIB
07	SYSLST	One-page preface to Lister option listing	LST
08	SYSLST SYSPCH	Detailed and summary listings Reformatted source deck	LST LST
10	SYSLST <sup>1,5</sup>	Source coding of Identification, Environ- ment, and Data Divisions (less Report Section)	LIST
12	SYSLST <sup>1,5</sup>	Source coding of Report Section	LIST
11	SYSLST <sup>1,5</sup>	Source coding of Procedure Division	LIST
25	SYS005	SYMDMP Dictionary (DATATAB)	SYMDMP <sup>2</sup>
30	SYSLST	Glossary	SYM
60 or 62,63,64 <sup>3</sup>	SYSLST SYSLST SYSPCH SYSLNK	Object module Object module (condensed) Object deck Object module	LISTX and NOCLIST <sup>4</sup> CLIST <sup>4</sup> DECK CATAL or LINK
65		Debugging information in object module or on SYS005	FLOW, STATE, or SYMDMP <sup>2</sup>
61	SYSLST	Cross-reference tables	SXREF, XREF, VERBSUM or VERBREF
70	SYSLST SYSLST	Error messages Error and warning messages	ERRS and FLAGE ERRS and FLAGW
80	SYSLST	Source program with FIPS flagging	LVL

<sup>1</sup>If LIB is in effect, source coding is listed by phase 01.  
<sup>2</sup>The FLOW, STATE, or SYMDMP option is required for phases 25 and 65.  
<sup>3</sup>The OPT option is required for phases 62, 63, and 64.  
<sup>4</sup>May be changed by SUPMAP option (see "Compiler Options" in this chapter).  
<sup>5</sup>If FIPS flagging has been requested (LVL option), source program is written on SYS006 for phase 80 input.  
<sup>6</sup>Only if LIB, NOLVL, and no Lister options.

Figure 2. Compiler Output

Text	Produced or Passed			Description
	By(Ph)	To(Ph)	On(File)	
IP	05	06	SYS002	Source program with syntactic markers inserted
IP	06	08	SYS002	Source program with cross-reference information included
Data IC	10	20	SYS003	I/O and data items from source program Environment and Data Divisions
P0	12	30	SYS002	Generated Report Writer subprogram from Report Section of source program Data Division
Data IC	12	20	SYS003	Data items from source program Report Section
P0	11	30	SYS002	Translation of source program Procedure Division
Data IC	20	22	SYS004	Translation of FDs and SDs by phase 10; passed unchanged
ATF	20	22	SYS004	Record Descriptions of levels 01-49, 66, 77, and 88. ATF-text is used as part of the dictionary entries
Data A (Incomplete)	20	22	SYS004	Constants defined by VALUE clauses. This text has a 2-byte prefix for identification, but lacks location fields
P0	22	30	SYS002	Generated Q-Routines from OCCURS...DEPENDING ON clauses
Data IC	22	21	SYS003	Translation of FDs and SDs by Phase 10; Passed unchanged
Data A	22	21	SYS003	Remainder of the source program Data Division. This text has a 2-byte prefix for identification
DEF	22	21	SYS003	Data-names and file-names from Data Division for use by phase 61 in producing the cross-reference listing. This DEF-text is put out by phase 22 when SXREF, XREF, VERBREF, VERBSUM, or SYMDMP has been specified. When SYMDMP has been specified, phase 21 copies the text onto SYS004. If SXREF, XREF, VERBREF, or VERBSUM has not been specified, phase 60 reads the text and ignores it.
FTL	22	21	SYS003	FDs and SDs

<sup>1</sup>Phase 60 is replaced by phases 62, 63, and 64 when the optimizer option (OPT) has been specified.

<sup>2</sup>Error text is treated somewhat differently. It is written on SYS003 for phase 70 by phase 60 or 64 only if the ERTBL table cannot accommodate it. It is produced by all phases between phase 01 and 51, and the accumulation is passed on SYS004. Phase 60 or 64 or phase 70 (if phase 60 or 64 text processing is bypassed or if SYNTAX is on) reads it from SYS004.

<sup>3</sup>When OPT is specified, file SYS004 is used for REF-text.

Figure 3. List of Internal Compiler Texts (Part 1 of 3)

Text	Produced or Passed			Description
	By(Ph)	To(Ph)	On(File)	
Data A	21	60 or 64 <sup>1</sup>	SYS004	Coding similar to assembler language for translation into data and global table portions of object module
DEF	21	60 or 64 <sup>1</sup>	SYS004	Data-names and file-names from Data Division. This DEF-text is put out by phase 22 when SXREF, XREF, VERBREF, VERBSUM, or SYMDMP has been specified. When SYMDMP has been specified, phase 21 copies the text onto SYS004. If SXREF, XREF, VEREREF, or VERBSUM has not been specified, phase 60 reads the text and ignores it.
P0	21	30	SYS002	VCONs for system modules, such as logic modules that will be required, determined from Data Division statements
DEF	30	60 or 64 <sup>1</sup>	SYS004	Procedure-names for use by phase 61 in producing the cross-reference listing
P1	30	40	SYS003	Further translation of P0-text. Names have been replaced by attributes; CORRESPONDING statements have been expanded
P2	40	50	SYS001	Further translation of P1-text. A more precise and expanded version of Procedure Division statements
Intermediate A	50	51	SYS002	Procedure A-text and Optimization A-text elements with an identifying prefix of X'27' or X'28'
Optimization A	50	60 or 62 <sup>1</sup>	SYS003	A special text, consisting of such items as virtuals and literals, used by phase 60 or phase 62 to eliminate storage duplication
Procedure A	51	60 or 62 and 63 <sup>1</sup>	SYS001	Coding similar to assembler language coding. It is ready for conversion into the object module
Optimization A	51	60 or 62 <sup>1</sup>	SYS003	A special text, consisting of such items as virtuals and literals, used by phase 60 or phase 62 to eliminate storage duplication
Procedure A1	63	64	SYS002	Coding produced from Procedure A-text, optimized to eliminate unnecessary Procedure Division instructions. It is similar to assembler language, and is ready for conversion into the object module
<sup>1</sup> Phase 60 is replaced by phases 62, 63, and 64 when the optimizer option (OPT) has been specified. <sup>2</sup> Error text is treated somewhat differently. It is written on SYS003 for phase 70 by phase 60 or 64 only if the ERRTBL table cannot accommodate it. It is produced by all phases between phase 01 and 51, and the accumulation is passed on SYS004. Phase 60 or 64 or phase 70 (if phase 60 or 64 text processing is bypassed or if SYNTAX is on) reads it from SYS004. <sup>3</sup> When OPT is specified, file SYS004 is used for REF-text.				

Figure 3. List of Internal Compiler Texts (Part 2 of 3)

Text	Produced or Passed			Description
	By(Ph)	To(Ph)	On(File)	
E	60 or 64 <sup>1</sup>	70	SYS003	Error Text <sup>2</sup>
E		70	SYS004	Error Text <sup>2</sup>
REF	60 or 64 <sup>1</sup>	61	SYS003 SYS004 <sup>3</sup>	For cross-reference table. This text is passed only if the SXREF, XREF, VERBREF, or VERBSUM option is in effect
DEF	60 or 64	61	SYS001	For cross-reference table. This text is passed only if the SXREF, XREF, VERBREF, or VERBSUM option is in effect
Debug	60	65	SYS002	For object-time debugging. This text is passed only if the SYMDMP or STATE option is in effect

<sup>1</sup>Phase 60 is replaced by phases 62, 63, and 64 when the optimizer option (OPT) has been specified.

<sup>2</sup>Error text is treated somewhat differently. It is written on SYS003 for phase 70 by phase 60 or 64 only if the ERRTBL table cannot accommodate it. It is produced by all phases between phase 01 and 51, and the accumulation is passed on SYS004. Phase 60 or 64 or phase 70 (if phase 60 or 64 text processing is bypassed or if SYNTAX is on) reads it from SYS004.

<sup>3</sup>When OPT is specified, file SYS004 is used for REF-text.

Figure 3. List of Internal Compiler Texts (Part 3 of 3)

GROUP 1: Control and Initialization

Phase 00 controls the flow. It receives control from and relinquishes it to the DOS system. It interfaces with the system for such Control Program services as input/output requested by other phases.

Phase 00 also serves as an interface between the other phases, controlling the flow and holding shared information in COMMON (see "Section 5. Data Areas"). After each phase, except phase 80, control returns to phase 00, which links to the next one; phase 80 returns control to the DOS/VS system. The Table Area Management Executive Routines (TAMER), part of Phase 00, allocate storage and process tables for the other routines. The "Table and Dictionary Handling" chapter describes the function of TAMER. For descriptions of tables handled by TAMER, and for the dictionary format, see "Section 5. Data Areas".

Phase 01 is logically a subroutine of phase 00 but does not remain in storage once its tasks are accomplished. This phase sets the options applicable for the compilation and opens all files.

If the BASIS or COPY function is used, the LIB option must be specified on the CBL

card. When LIB is specified, Phase 01 reads the source program and resolves BASIS and COPY statements by performing syntax analysis on these statements and by writing the user created COBOL libraries inline with the remainder of the source program on SYS004. E-text for BASIS and COPY functions is written on SYS003 and the entire source language listing is produced on SYSLST (or SYS006 for LVL option).

GROUP 2: Reformatting the Source Deck

If the LST option is specified phases 05, 06, 07, and 08 produce a reformatted source program with embedded cross-reference information and a summary of cross-reference information on SYSLST. Optionally they also produce a reformatted source deck on SYSPCH. These phases also write the source program on SYS004.

GROUP 3: Translating the Identification, Environment, and Data Divisions

Phases 10, 20, 22, and 21 process the Identification, Environment, and Data Divisions, with the exception of the Report

Section, if any, (discussed with Group 3, below). Those source statements requiring translation are changed to Data IC-text in phase 10. Each of the other three phases translates a portion of this Data IC-text into Data A-text. The result is later joined to the Procedure Division by Phase 60 or to Phase 64 (Group 4, below) if the optimizer option has been specified.

DEF-text, which provides references to data-names and file-names required for

cross-reference listings, is created in phase 22 and passed through phase 21 to phase 60 or phase 64. Phase 22 also enters complete entries for LDs and RDs (Record-level descriptions) and partial (dummy) entries for SDs and FDs in the dictionary. The SD and FD entries are completed by phase 21, and the dictionary is then ready for use by phase 30 and by phase 25 (if the SYMDMP option is in effect). P0-text, the first form of internal procedure text (see Group 3





below), is created by phase 22 for Q-Routines and by phase 21 for VCONS. Q-Routines are special routines created to handle OCCURS clauses with the DEPENDING ON option. They are explained further in the chapter on phase 22. The P0-text is placed on SYS002 for later use by phase 30. Finally, if the LIST option is in effect and the LST option is not in effect, phase 10 produces a source language listing of the Identification, Environment, and Data Divisions on SYSLST (or SYS006 for LVL option). (If LIB was specified, the source listing is produced by Phase 01.)

Each of these phases scans its input text for errors and adds any necessary diagnostics, written in E-text, to the textual flow. The accumulated E-text is intermingled with the Data A-text and DEF-text on SYS004, and all three provide part of the input for Phase 60 or Phase 64. If a listing of error messages is to be produced, Phase 70 is later called in to produce it.

The translation process can be visualized from the contents of the work files produced by the phases. These work files are shown in Figure 4. Note that, in addition to the contents shown, the work files contain E-text and DEF-text. The texts are intermingled on the work files, but they are readily distinguishable by their prefixes. The formats of the texts are shown in "Section 5. Data Areas."

#### GROUP 4: Translating the Procedure Division

If the Report Writer facility is required, phase 00 calls on phase 12. This phase reads the Report Section of the Data Division, generates the report program in P0-text, and writes it on SYS002. Phases 11, 30, 40, 50, 51, and 52 then process the Procedure Division.

Phase 11 encodes the source program's Procedure Division into P0-text, and adds it to SYS002. It also begins the dictionary by placing procedure-names and their attributes in it. SYS002 and the dictionary, with the additions from phases 22 and 21, later provide input for phase 30.

Phase 30 uses the information in the dictionary to replace each name with its attributes. It performs any other processing requiring the dictionary and then releases the dictionary's storage space. It also translates the P0-text into P1-text. The text is changed into P2-text by phase 40, and into Procedure A- and Optimization A-texts by phases 50, 51, and

52. These last two texts are placed on SYS001 and SYS003 for phase 60 or phase 62.

Phase 60 converts Procedure A-text into the machine language program. When the optimizer option (specified by OPT on the CBL card) is in effect, phases 62 and 63 read Procedure A-text and phase 63 converts it into Procedure A1-text and passes it to phase 64. Phase 64 then converts Procedure A1-text into the machine language program.

Phase 11 also writes a source listing of the Procedure Division on SYSLST (SYS006 for LVL option) if the LIST option is in effect. (If LIB was specified, Phase 01 writes the source listing.) Phase 30 writes a glossary on SYSLST (SYS006 for LVL option) if the SYM option is in effect. Phase 30 also creates DEF-text for procedure-names and places it on SYS004 for later use in producing the cross-reference listing.

Each of these phases scans its input text for errors and adds any necessary diagnostics, written in E-text, to the textual flow. The accumulated E-text is intermingled with the internal procedure text until it is isolated by Phase 51 and added to SYS004 for phase 60 or phase 64.

#### GROUP 5: Assembly

The function of phase 60 or of phases 62, 63, and 64 is to produce from the texts, tables, and counters that have been created by the earlier phases a machine language program that is suitable for input to the linkage editor. Phases 62, 63, and 64 are an optional version of phase 60 and are used instead of phase 60 when the optimizer option (OPT) is requested.

Phase 60 produces code that has been optimized for literals, virtuals, source procedure-names, and compiler-generated procedure-names. Phases 62, 63, and 64 produce code that has been optimized for instructions generated from the Procedure Division as well as those items optimized for by phase 60. To produce the object module, the Data A-text on SYS004, the Procedure A-text on SYS001 (used by phase 60) or the Procedure A1-text on SYS002 (used by phase 64), and the Optimization A-text on SYS003 are used. Phase 63 creates Procedure A1-text from Procedure A-text.

The object module is written, according to the options that are in effect, as an object deck on SYSPCH if DECK is in effect, as an object listing on SYSLST (SYS006 for LVL option) if LISTX is in effect, as input

Work File	Produced By	Read By	Contents
SYS003	Phases 10 & 12	Phase 20	Data Division, in Data IC-text
SYS004	Phase 20	Phase 22	<ol style="list-style-type: none"> <li>1. FDs and SDs, in Data IC-text</li> <li>2. Objects of RENAMEs clauses, in Data IC-text</li> <li>3. Table handling keys, in Data IC-text</li> <li>4. Constant definitions in incomplete Data A-text</li> <li>5. Remainder of Data Division, in ATF-text</li> </ol>
SYS003	Phase 22	Phase 21	<ol style="list-style-type: none"> <li>1. FDs and SDs, in Data IC-text</li> <li>2. Remainder of Data Division, in Data A-text</li> </ol>
SYS004	Phase 21	Phase 60 or 64	Data Division in Data A-text.

Figure 4. Contents of Work Files Used in Translating the Data Division

to the Linkage Editor on SYSLNK if either LINK or CATAL is in effect, and as a condensed object listing on SYSLST (SYS006 for LVL option) if CLIST is in effect.

GROUP 6: Final and Diagnostic Output

Final and diagnostic output for the compiler consists of the object module discussed above and other optional information.

If the SXREF or the XREF option is in effect, phase 60 or phase 64 reads the DEF-text from SYS004 and places it on SYS001. Phase 60 or 64 writes the necessary REF-text, placing it on SYS003 or SYS002, respectively. phase 00 calls Phase 61, an optional phase, which produces a cross-reference listing on SYSLST (or SYS006 for LVL option).

If the SYMDMP option is in effect, phase 25 is called to create tables in the Debug File used by the object-time COBOL library subroutines in producing the formatted dump. If the SYMDMP, STATE, or FLOW option is in effect, phase 60 produces debugging information and passes it to phase 65, an optional phase. For SYMDMP, phase 65 completes the Debug File; for STATE, it uses the debugging information written by phase 60 to produce two tables in the object module; for FLOW, it places the number of traces requested in the variable

portion of the Task Global Table in the object module and allocates space after INIT3 for the table used by the COBOL library object-time subroutine ILBDFLW0. For any of these options, phase 65 writes the end card in the object module, writes information in the TGT, and does the processing necessary for segmented programs and programs that use the SORT verb or MERGE verb.

Phase 70 produces an error listing on SYSLST (SYS006 for LVL option) and is only given control if source program errors were detected. Phase 00 calls phase 70 and the input to the phase in E-text. Any phase which found an error produced an E-text element, specifying the error message to be written. These E-text elements are collected by phase 60 or phase 64 and used to produce the list of error messages (and warning messages if the user requested them).

Phase 80 scans the source program for deviations from the Federal Information Processing Standard (FIPS) and issues messages along with the source program listing.

COMPILER OPTIONS

Phase 01 stores information in COMMON in accordance with the options which are to control the compilation step.

All of the following options except LINECT may be set by the user at installation time as defaults or by the programmer on the OPTION control card. LINECT may be set at installation time as a default or by the operator.

If no Lister options are in effect and an option causes output to be written on SYSLST, the output is written directly on SYSLST, if the NOLVL option is in effect. If the LVL option is in effect, the output is written on SYS006, which is used by phase 80 to produce a listing on SYSLST.

CATAL The object module is placed on SYSLNK and catalogued in the core image library after link editing.

DECK or NODECK A deck is produced on SYSPCH.

DUMP or NODUMP A listing of registers and storage is printed when an abnormal program termination occurs.

ERRS or NOERRS Compiler diagnostics are printed on SYSLST.

LINECT=nn The specified number of lines ('nn') are printed on each listing page.

LINK or NOLINK The object module is placed on SYSLNK.

LIST or NOLIST The source module is printed on SYSLST. If LST is in effect, NOLIST is ignored.

LISTX or NOLISTX The object module is printed on SYSLST.

LOG or NOLOG A listing of all control statements is printed.

SYM or NOSYM Global tables, literal pools, register assignments, and glossary listing are printed on SYSLST.

XREF or NOXREF A cross-reference listing is printed on SYSLST.

APOST or QUOTE

The apostrophe or the double quotation mark has been used to delineate literals or represents the character used when the figurative constant QUOTE is used.

BUF=nnnnn

Each compiler work file buffer is assigned 'nnnnn' bytes of storage. The minimum is 512 bytes, and the maximum is the maximum block size for the storage device used. The default is 512 bytes.

CATALR or NOCATALR

CATALR indicates that CATALR card images are to be written on the SYSPCH file if DECK is specified on the OPTION card. This allows the object modules produced by the compiler to be catalogued in the relocatable library. The module names on the CATALR cards follow the same rules as the phase names in the compiler produced PHASE cards according to the segmentation and sort phase naming conventions.

CLIST or NOCLIST

Global tables, literal pools, register assignments, and a condensed listing are produced. If OPT is specified, the starting address of each Procedure Block (PBL) is also listed. The procedure portion contains source card numbers and the relative location of the first generated instruction for each verb. This option overrides NOLISTX. The default is LISTX.

COUNT or NOCOUNT

COUNT indicates that an execution summary is to be produced at the end of execution of the compiled program. If both COUNT and STXIT are desired, the program unit requesting COUNT either must contain the request for STXIT or

The following options may be set by the programmer using the CBL card or by making appropriate entries in source library member C.CBLOPTNS. Any number of CBL cards may be used. The default cases are underlined.

	must be entered before the program unit that requests STXIT. When COUNT is in effect, the maximum number of verb blocks is 32,767.	PMAP=n	This option enables the user to request a relocation factor "n" to be added to the location counter field in the object code listing. The relocation factor "n" is a hexadecimal number of from one to eight digits. If the PMAP option is not specified, the relocation factor is assumed to be zero. If the PMAP option is specified in a segmented program, the object code listing for segments of priority higher than the segment limit (default = 49) will not be relocated. This option is in effect only if LISTX is specified.
<u>FLAGW</u> or <u>FLAGE</u>	All warning and error diagnostics ( <u>FLAGW</u> ) or only error diagnostics ( <u>FLAGE</u> ) are listed.		
FLOW =(n[ <u>n</u> ])	A formatted trace (i.e., a list containing the program identification and statement numbers) corresponding to a variable number of procedures executed prior to an abnormal termination is printed on SYSLST. FLOW must not be specified for the same compilation as OPT or STXIT.		
<u>LIB</u> or <u>NOLIB</u>	LIB indicates that BASIS and/or COPY statements are in the source program. If either COPY or BASIS is present, LIB must be in effect. If neither COPY nor BASIS statements are present, use of the NOLIB option yields more efficient compiler processing.	<u>SEQ</u> or <u>NONSEQ</u>	The source statements are ( <u>SEQ</u> ) or are not ( <u>NONSEQ</u> ) sequence checked. If LST is in effect, this option is ignored.
<u>LVL=c</u> <u>NOLVL</u>	Indicates whether the Federal Information Processing Standard 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). When LVL=c is designated as the default at installation time, NOLVL can not be specified at compile-time.	<u>SXREF</u> or <u>NOSXREF</u>	An alphabetically ordered cross-reference listing is printed on SYSLST. SXREF overrides XREF, which is specified on the OPTION control card. The default value is XREF.
		<u>SPACE1</u> , <u>SPACE2</u> , or <u>SPACE3</u>	The output listing is single, double, or triple spaced.
		<u>STATE</u> or <u>NOSTATE</u>	The number of the statement being executed at the time of abnormal termination is printed on SYSLST. STATE must not be specified for the same compilation as SYMDMP, OPT or STXIT.
<u>OPT</u> or <u>NOOPT</u>	The object module is optimized by Phases 62, 63, and 64 for instructions generated from the Procedure Division. If OPT is specified, the starting address of each Procedure Block (PBL) is also listed. OPT must not be specified for the same compilation as SYMDMP, FLOW, or STATE.	<u>STXIT</u> or <u>NOSTXIT</u>	Control is passed to a user error procedure if an error occurs on a unit-record file. If both COUNT and STXIT are desired, the program unit requesting COUNT either must contain the request for STXIT or must be entered before the program unit that requests STXIT.

SUPMAP or  
NCSUPMAP

The Phase 60 output is suppressed if any E-level messages are produced by the compiler.

SYMDMP[=filename]

A formatted symbolic dump of specified data areas is printed on SYSLST at various points dynamically during execution of a program; in the event of an abnormal termination, a formatted symbolic dump

of all data areas is printed on SYSLST. SYMDMP must not be specified for the same compilation as STATE, OPT or STXIT. For all the mutually exclusive options, the last encountered option is the one remaining in effect. If "filename" is not specified, the debug file (on SYS005) is named IJSYS05. If more than one COBOL



<p>SYNTAX          CSYNTAX  <u>NOSYNTAX</u></p>	<p>program with the SYMDMP option is included in a run unit and all the debug files are on the same direct access device, each must be given a unique name.</p>	<p>SXREF, and LISTX are not specified</p>
	<p>Indicates whether the source text is to be scanned for syntax errors only and appropriate error messages are to be generated. For conditional syntax checking (CSYNTAX), a full compilation is produced as long as no messages exceed the W level. If one or more C-level or higher severity messages are produced, the compiler generates the messages but does not generate object text.</p>	<p><u>TRUNC</u> or NOTRUNC Standard or nonstandard truncation will be applied to computational items. If standard (TRUNC), items are truncated according to their pictures; if nonstandard (NOTRUNC), they are truncated according to the actual amount of storage occupied.</p>
	<p><u>Notes:</u></p>	<p>VERB or <u>NOVERB</u> 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 LISTX or CLIST compiler option is specified, if READY TRACE is used in the source program, or if VERBSUM, VERBREF, or COUNT is specified.</p>
	<p>1. When the SYNTAX option is in effect, all of the following compile-time options are suppressed:</p>	<p>VERBREF or NOVERBREF VERBREF indicates that a verb cross-reference listing is to be produced for the compiled program.</p>
	<p>OPTION control statement: LINK, DECK, XREF</p>	<p>VERBSUM or NOVERBSUM VERBSUM indicates that a verb summary listing is to be produced for the compiled program.</p>
	<p>CBL statement: SXREF, CLIST, LISTX, VERBSUM, VERBREF, COUNT</p>	<p><u>ZWB</u> or NOZWB Indicates whether the compiler is to generate code to strip the sign when a signed external decimal field is compared to an alphanumeric field. ZWB specifies stripping.</p>
	<p>2. When conditional syntax-checking is requested, the preceding options are suppressed only if one or more E- or D-level messages are generated.</p>	<p>The following Lister options may be set by the programmer using the LST card. The default cases are underlined.</p>
	<p>3. Unconditional syntax checking is assumed if all of the following compile-time options are specified:</p>	<p><u>COPYPCH</u> or <u>NOCOPYPCH</u> COPYPCH indicates that the updated and reformatted copy libraries are to be punched. If the Lister DECK option is in effect, the libraries will be punched as part of the source deck. If the Lister NODECK option is in effect, the libraries will be</p>
	<p>OPTION control statement: NOLINK, NOXREF, NODECK</p>	
	<p>CBL statement: SUPMAP, CLIST, VERBSUM, VERBREF,</p>	

	punched as a separate deck.		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.
DECK or <u>NODECK</u>	DECK indicates that an updated source deck is to be produced. If the COPYPCH option is in effect, the updated source deck will include the updated and reformatted copy libraries.	<u>PROC=1COL</u> or <u>PROC=2COL</u>	PROC=1COL indicates that the Procedure Division is to be listed in single-column format. PROC=2COL, which can be specified only if the printer has 132 print positions, indicates that the Procedure Division is to be listed in double-column format.
<u>LSTONLY</u> or <u>LSTCOMP</u>	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 Lister DECK option is in effect		



SECTION 2. METHOD OF OPERATION

PHASE 00

Phase 00 (ILACBL00), the interface between the COBOL compiler and the DOS/VS System, is resident in storage throughout compilation. Its major functions are:

- Receiving control from the DOS/VS System and, at the end of compilation, returning control to it.
- Performing reallocation and linkage after each of the other phases has completed its operations.
- Handling input/output requests from the other phases.
- Manipulating tables for the other phases.
- Providing a communications area (COMMON) for the other phases.
- Processing terminal error conditions.

RECEIVING CONTROL FROM AND RETURNING IT TO THE SYSTEM

Compilation is invoked by an EXEC control card. Phase 00 is entered at entry point START. Routine LINKA calls phase 01, and passes to it the parameters for processing and the addresses of certain areas in phase 00 for phase 01 to fill in.

When phase 60, 64, 65, 61, or 70 is the last phase, it calls phase 00 after its processing is complete. Routine SKPLNK issues an EOJ macro to return control to the DOS/VS System. If an error occurs that stops compilation (see "Unrecoverable Error Conditions" in this chapter), routine TRMNATE prints a message and returns control to the system with a CANCEL or DUMP macro instruction. If an error occurs and OPT is specified, phase 62 returns to phase 00 with a CANCEL code. When LVL is in effect, phase 80 is the final phase and it returns control directly to the system.

Phase 00 keeps track of which processing phase is currently active by means of a

two-byte cell named LINKCNT. Routine LINKA increments LINKCNT by two before it links to the next phase, so that, for example, the value of LINKCNT is four at entry to phase 10. If phase 70 is not executed, or if termination is abnormal, the termination routines set LINKCNT to the value it would have on entry to phase 70, that is, the hexadecimal value '26'. The location of LINKCNT is internal to phase 00.

Figure 4 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 5 for 'XY' values.
3. 'ZZ' indicates functions to be performed by phase 00. See Figure 5 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."



X Code	Routine Called	Function of Routine
0	READ	Read a utility file. Pass back to the calling routine the storage address of the logical record. If Y is 9, read a COPY or BASIS library.
1	WRITE	PUTN: Write IC-text, where the calling routine gives the record's address in register 2, and its length in register 3.
2	WRITEA	PUT: Write a record, where the calling routine gives the record's address in register 2, and the first two bytes of the record define its length.
3	TRMNATE	Cancel job. A D-level error has been encountered.
6	CLOSET	Purge, if necessary, and rewind the indicated file. If the second parameter byte (ZZ) contains X'00', last use of file was to read. If the second parameter byte (ZZ) contains X'01', last use of file was to write.
7	READQ	Read SYSIPT.
9	SEGPNT	Position the access mechanism to a disk address supplied by the calling routine (on SYS001) and read the record (see phase 60 and phases 62 and 63).
A	LINKB	Issue a RETURN macro to terminate the previous phase.
B	EOJ	Return to the system.
C	SEGNOTE	Write the current record, if required, in a short block, and pass back to the calling routine the relative disk address of the next record to be written on SYS001 (see phase 51).
D	EJECT or SKIP	Position the printer. The exact function is determined by a second parameter byte as follows:  X'00' Eject X'01' Skip one line X'02' Skip two lines X'03' Skip three lines
E	CLOSER	Flush the buffer. Do not write an end-of-file indicator (see F). If the second parameter byte (ZZ) contains X'CC', call was not from phase 00 internally but from another phase.
F	CLOSER	When a file is to be closed, move 'FF' and an end-of-file code into the buffer, pad the buffer with zeros, check the previous input/output operation, and write out the file.

Y Code	File	ZZ Code	Meaning
0	SYS005	X'01'	NOTE macro instruction to retrieve the absolute address
1	SYS001	X'02'	POINT macro instruction to cause processing to start at the specified block in the file
2	SYS002		
3	SYS003	X'03'	POINTS to rewind the file
4	SYS004	X'04'	WRITE UPDATE (disk only)
5	SYSIPT	X'05'	When preceded by a X'22', WRITE on SYS002 from SYS005
6 <sup>1</sup>	SYSLST		buffer with a buffer size of 512 bytes; when preceded by a
7	SYSPCH		X'02', READ from SYS002 into SYS005 buffer with a buffer
8	SYSLNK		size of 512 bytes.
9	SYSSLB		

<sup>1</sup>Also for SYS006 when LVL is in effect.

Figure 5. Linkage Codes to Phase 00

When the linkage code indicates end of phase the following occurs:

- Routine LINKB issues RETURN macro to terminate calling phase.
- Control passes to the instruction following the LINK instruction in routine LINKA.
- Files are purged.
- If necessary, an interlude routine is called (INT1 after phase 10; INT11 after phase 11, etc.). The interlude routine rewinds files, sets up POINT table buffers, determines next phase, etc.
- Routine LINKA updates LNKCNT, sets switches for TAMER, loads and calls the next phase.

Figure 6 shows the conditions under which optional phases are called. Figure 8 shows the flow of control for processing between phases.

Use of buffers by files in each phase is predetermined and is described in Figure 10. Buffer addresses are recorded in a buffer pointer table. Phase 00 contains a pointer to the current buffer address.

Phase 01 opens all files.

Phase 00 closes work files as they are no longer needed.

Optional Phase	Preceding Phase	Compiler Option
05	01	LST
06	05	LST
07	06	LST
08	07	LST
25	21	SYMDMP
61	60 64 65	XREF SXREF VERBSUM VERBREF
62 63 64	51	OPT
65	60 64	SYMDMP
70	Varies	If errors in source program
80	60 61 64 65 70	LVL

Figure 6. Optional Phase Processing

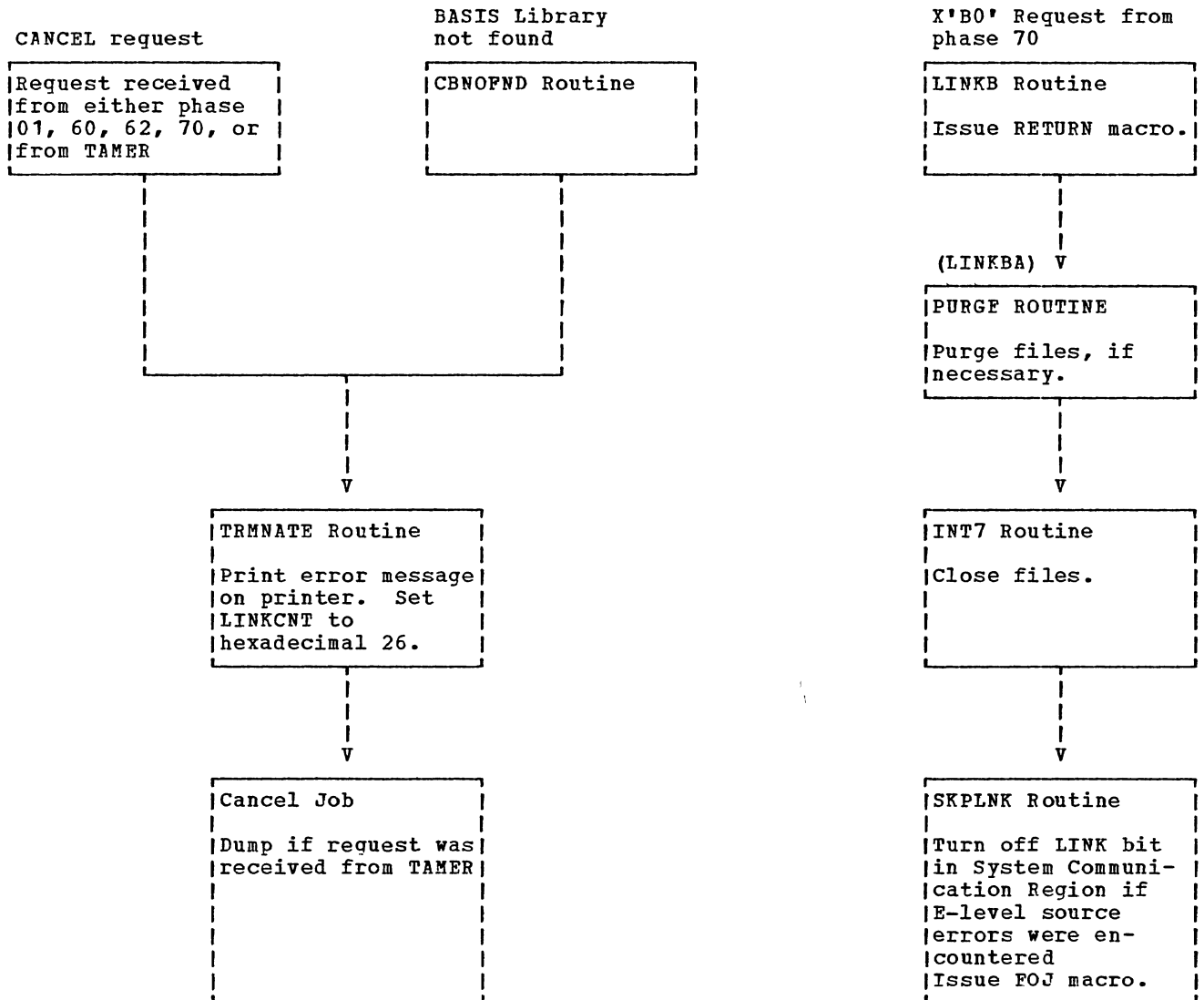
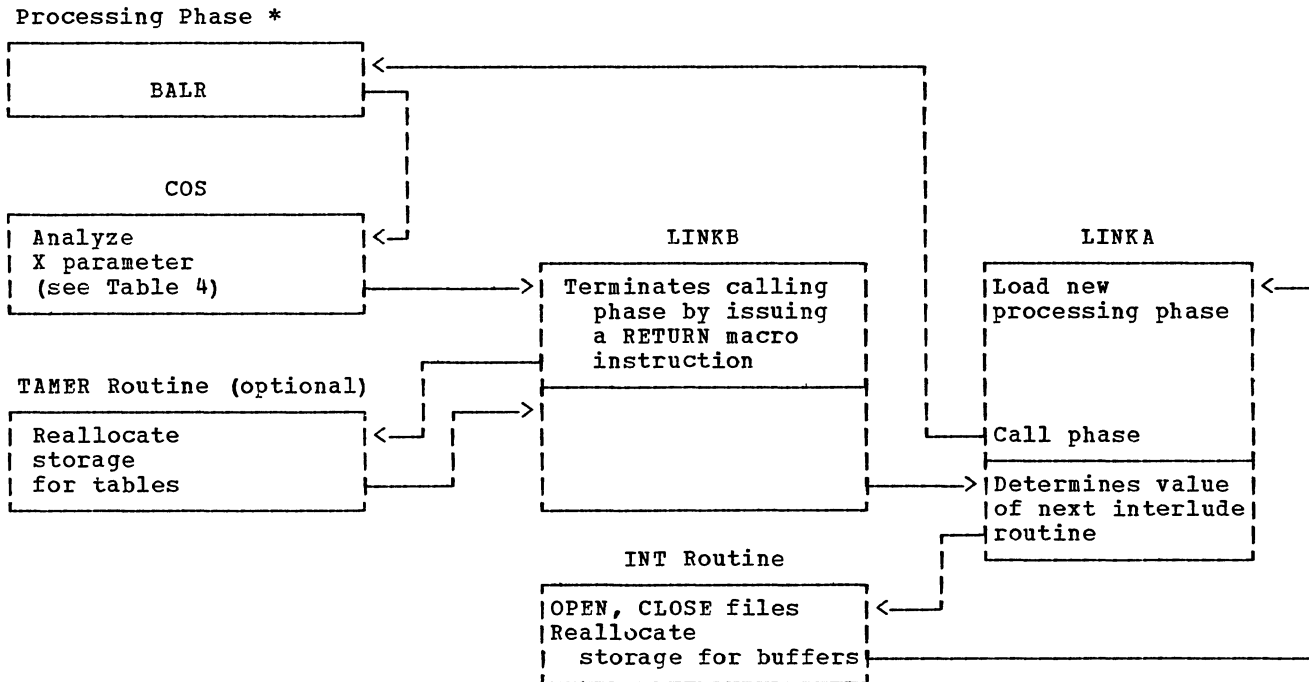


Figure 7. Flow of Control at End of Compilation (Normal and Abnormal)

PHASE INPUT/OUTPUT REQUESTS

Phase 00 translates all phase input/output requests into system macro instructions or SVCs. It switches the buffer pointers in the POINT table if the

file is double buffered, and checks to determine whether the operation was completed successfully. If necessary, TAMER handles dictionary spill during phase processing (see "Appendix A. Table and Dictionary Handling"). Figure 9 shows the input/output requests for each phase.



Note: The blocks do not indicate relative locations or sizes of the routines.

\*This box changes contents at the end of each cycle. Exit from the cycle occurs after the last processing phase, which may be phase 60, 61, 62, 64, 65, or 70, when control is returned to COS with a request for an EOJ macro instruction. Exit may occur through abnormal termination (see Figure 7).

Figure 8. Flow of Control for Processing Between Phases

Phase	SYS001	SYS002	SYS003	SYS004	SYS005	SYSIPT	SYSLST <sup>18</sup>	SYSPCH	SYSLNK	SYSSLB
01	Open	Open	Open E-text <sup>16</sup>	Open Put <sup>16</sup>	Open	Open Get <sup>16</sup>	Open Put <sup>16</sup> Close <sup>21</sup>	Open	Open	Open Get Close
00 (INT01)				Rewind						
05 (if LST)		Write		Get <sup>25</sup>		Get				
00 (INT05)		Flush Rewind		Rewind						
06 (if LST)		Read Write		Write Read						
00 (INT06)		Flush Rewind		Flush Rewind						
07 (if LST)							Put			
00 (INT07)										
08 (if LST)		Read		Write			Put Close <sup>24</sup>	Put		
00 (INT08)		Rewind		Flush Rewind						
10			Write Data IC-text & E-text	Get <sup>16, 22</sup>		Get <sup>17, 23</sup>	Put <sup>17, 23</sup>			
00 (INT1)										
12 (Report writer)		Write PO-text & E-text	Write Data- IC-text	Get <sup>16, 22</sup>		Get <sup>17, 23</sup>	Put <sup>17, 23</sup>			
00 (INT12)										

Footnotes. Only those footnotes referred to on this page are listed below.

- <sup>1</sup>Used for dictionary spill.
- <sup>16</sup>If LIB or LST is in effect.
- <sup>17</sup>If NOLIB is in effect.
- <sup>18</sup>If LVL is in effect, SYS006 is used in place of SYSLST.
- <sup>21</sup>If LVL is in effect and no Lister options, SYS006 is opened for LVL.
- <sup>22</sup>If LSTCOMP is in effect.
- <sup>23</sup>If LSTCOMP is not in effect.
- <sup>24</sup>Close SYSLST if LVL and LSTCOMP are in effect; SYS006 is used for LVL.
- <sup>25</sup>Only if LIB is in effect; otherwise input is on SYSIPT.

Figure 9. Compiler File Handling (Part 1 of 6)

Phase	SYS001	SYS002	SYS003	SYS004	SYS005	SYSIPT	SYSLST <sup>18</sup>	SYSPCH	SYSLNK	SYSSLB
11	(See note 1)	Write P0-Text & E-Text		Get <sup>16, 22</sup>		Get <sup>17, 23</sup>	Put <sup>17, 23</sup>			
00 (INT11)		Flush	Flush Rewind			Close				Close
20			Read Data IC- and E-texts	Write Incomplete Data A-text, Data IC-text, ATF-text, & E-text						
00 (INT2)			Rewind	Rewind						
22		Write P0-text (Q-routine)	Write Data A- & Data IC-texts DEF-text <sup>8</sup>	Read Data IC-text, Incomplete Data A-text, ATF-text and E-text						
00 (INT22)		Flush	Flush Rewind	Rewind						
21	(See note 1)	Write P0-text (VCONS)	Read Data IC-text Data A-text, & DEF-text <sup>8</sup>	Write Data A-text E-text, & DEF-text <sup>8</sup>						
00 (INT21)		Flush Rewind	Rewind	Flush						
25 <sup>7</sup>		Write E-text		Note Rewind Read DEF-text Point	Write DATATAB OBODOTAB					
00 (INT25)										
30	(See note 1)	Read P0 text (Q-rout) & E-text	Write P1-text & E-text	Write DEF-text <sup>2</sup>			Put (SYM)			
00 (INT3)	Rewind	Rewind	Flush Rewind	Flush						

Footnotes. Only those footnotes referred to on this page are listed below.

- <sup>1</sup>Used for dictionary spill.
- <sup>2</sup>SXREF, XREF, VERBSUM, or VERBREF only.
- <sup>7</sup>SYMDMP only.
- <sup>8</sup>SXREF, XREF, VERBSUM, VERBREF, or SYMDMP only.
- <sup>18</sup>If LVL is in effect, SYS006 is used in place of SYSLST.
- <sup>22</sup>If LSTCOMP is in effect.
- <sup>23</sup>If LSTCOMP is not in effect.

Figure 9. Compiler File Handling (Part 2 of 6)



Phase	SYS001	SYS002	SYS003	SYS004	SYS005	SYSIPT	SYSLST <sup>18</sup>	SYSPCH	SYSLNK	SYSSLB
40	Write P2-text & E-text		Read P1-text &E-text	Write E-text <sup>19</sup> &Data A-text <sup>20</sup>						
00 (Int4)	Flush Rewind		Rewind	Flush						
50	Read P2-text & E-text	Write P2-text, Inter- mediate Proced. A-text, Inter- mediate E-text & Inter- mediate Optimiz. A-text	Write Opt. A-text							
00 (INT5)	Rewind	Flush Rewind	Flush							
51	Write Proced. A-text	Read P2-text, Inter- mediate Proced. A-text, Inter- mediate E-text, & Inter- mediate Optimiz. A-text	Write Opt. A-text	Write E-text						

Footnotes. Only those footnotes referred to on this page are listed below.

<sup>18</sup>If LVL is in effect, SYS006 is used in place of SYSLST.

<sup>19</sup>If SYNTAX or CSYNTAX is in effect.

<sup>20</sup>If COUNT is in effect.

Figure 9. Compiler File Handling (Part 3 of 6)

Phase	SYS001	SYS002	SYS003	SYS004	SYS005	SYSIPT	SYSLST <sup>18</sup>	SYSPCH	SYSLNK	SYSSLB
00 (INT51)	Flush Rewind	Rewind <sup>5</sup>	Flush Rewind	Flush Rewind  No Rewind <sup>3</sup>						
60 <sup>13</sup>	Read Proced. A-text  Rewind <sup>2</sup>  Write DEF-text <sup>2</sup>	Write Debug- text <sup>5</sup>	Read Optimiz. A-text  Rewind <sup>2</sup>  Write REF- text <sup>2</sup> & E-text	Write SYSLNK OUTPUT for non-root segments <sup>3</sup>  Rewind <sup>3</sup>  Read Data A-, DEF-, and E-texts; Reads SYSLNK output for non-root segments <sup>3</sup>			Put (LISTX)	Put	Put	
00 (INT60) <sup>9</sup>	Rewind <sup>2</sup>	Flush Rewind <sup>5</sup>	Flush Rewind	Flush <sup>3</sup> No rewind <sup>15</sup> Rewind						
65 <sup>13</sup>		Read Debug- text Write text from SYS005 <sup>11</sup>		Read non-root segments	Write PROCTAB PROGSUM <sup>10</sup>			PUT	PUT	
00 (INT65)	Rewind <sup>2</sup>									
62 <sup>12</sup>	Read Proced. A Text		Read Optimiz. A-text							

Footnotes. Only those footnotes referred to on this page are listed below.

<sup>2</sup>SXREF, XREF, VERBSUM, or VERBREF

<sup>3</sup>Segmentation only.

<sup>5</sup>If SYMDMP or STATE in effect.

<sup>9</sup>Debugging options in effect. This phase can only follow phase 60.

<sup>10</sup>If SYMDMP is not in effect, SYS005 is not used.

<sup>11</sup>Use this file, if SYS005 is assigned to a tape device.

<sup>12</sup>Control flows in this sequence when the OPT option is in effect.

<sup>13</sup>Control flows in this sequence when the OPT option is not in effect. Phase 65 is an optional phase.

<sup>15</sup>If SYMDMP, STATE, FLOW with segmentation.

<sup>18</sup>If LVL is in effect, SYS006 is used in place of SYSLST.

Figure 9. Compiler File Handling (Part 4 of 6)

Phase	SYS001	SYS002	SYS003	SYS004	SYS005	SYSIPT	SYSLST <sup>1a</sup>	SYSPCH	SYSLNK	SYSSLB
00 (INT62)	Rewind	Rewind								
63 <sup>12</sup>	Read Proced. A-text	Write Proced. A1-text								
00 (INT63)	Rewind <sup>2</sup>	Flush Rewind	Rewind	Rewind						
64 <sup>12</sup>	Write DEF-text <sup>2</sup>	Read Proced. A1-text	Write E-text	Read DATA A- text, DEF- text, & E-text Rewind <sup>2</sup> Write REF-text <sup>2</sup>						
00 (INT64) <sup>6</sup>			Flush Rewind							
00 (INT6X) <sup>2</sup>	Flush <sup>2</sup> Rewind <sup>2</sup>		Flush Rewind	Flush <sup>2</sup> Rewind <sup>2</sup>						
61 <sup>2</sup>	Read DEF-text		Read REF- text <sup>7</sup>	Read REF- text <sup>12</sup>			Put (SXREF or XREF)			
00 (INT61) <sup>2</sup>				Rewind						
70			Read E-text	(See note 4)			Put (error messages)			
00 (INT7)	Close	Close	Close	Close			Close	Close	Close	

Footnotes. Only those footnotes referred to on this page are listed below.

<sup>2</sup>SXREF, XREF, VERBSUM, or VERBREF only.

<sup>4</sup>E-text read from SYS004 if phase 60 or 62 determines that no output is required or if SYNTAX is on.

<sup>6</sup>SXREF or XREF not in effect.

<sup>7</sup>SYMDMP only.

<sup>12</sup>Control flows in this sequence when the OPT option is in effect.

<sup>1a</sup>If LVL is in effect, SYS006 is used in place of SYSLST.

Figure 9. Compiler File Handling (Part 5 of 6)

Phase	SYS001	SYS002	SYS003	SYS004	SYS005	SYS006	SYSLST	SYSPCH	SYSLNK	SYSSLB
80 (only if LVL is in effect)	Open Write Read Close					Open Read Close	Open Write Close			

Figure 9. Compiler File Handling (Part 6 of 6)

TABLE AND DICTIONARY HANDLING

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

1. A permanent input/output error is encountered on a device.
2. An invalid BASIS library name is encountered.
3. The partition size is too small for this program.

Conditions detected by TAMER:

1. A table has exceeded the maximum permissible size.
2. Because of an error in compiler logic or a machine error, TAMER is unable to handle a request.

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 "Section 5. Data Areas."

SEGMENTATION OPERATIONS

For a segmented program phase 51 keeps track of the sections of Procedure A-text that belong to each segment. It calls the NOTE routine of phase 00 ('XY' code = C1) to record the absolute track address on SYS001 of the first record of each segment.

UNRECOVERABLE ERROR CONDITIONS

The following conditions will cause abandonment of compilation. In each case a console message is printed, and phase 00 returns control to the DOS/VS System via the routines described earlier in this chapter under "Receiving Control From and Returning It to the System."

Phase 60 or phases 62 or 63 use the SEGTBL table built by phase 51 to read Procedure A-text in order of ascending segment priority. They call the SEGPNT routine of phase 00 ('XY' code = 91) for this purpose. Phase 60 writes the object module and phase 63 writes Procedure A1-text in order of ascending priority.

Phase 01 (ILACBL01) initializes compiler operations. Although logically a subset of phase 00, it is not needed in storage throughout compilation and exists as a separate load module. The major functions of phase 01 are:

- Attempts to read source statement library member C.CBLOPTNS and, if present, sets the compiler default options to correspond to those in the library member. Normal default options are used if no library member is present.
- Determines compilation parameters from the system OPTION card and the COBOL compiler CBL and LST cards from SYSIPT and sets indicator bits in COMMON.
- Determines buffer sizes for all files used by the compiler.
- Obtains storage for buffers, tables, and the dictionary.
- Opens all files used by the compiler.
- Returns information to phase 00 on the results of the initialization.
- Takes appropriate action in the event of certain error conditions.
- Handles COPY and BASIS functions if LIB is specified.

#### COMPILATION PARAMETERS

Compilation parameters are set as defaults at installation time. Thereafter, they may be changed by the operator through a console command or by the programmer through a system OPTION card, one or more CBL option cards, or an LST option card. CBL option cards are placed in front of the first compiler source card.

Based on the parameters, phase 01 moves flag bytes to the LISTERSW, PHZSW, PHZSW1, PHZSW2, PHZSW3, and PHZSW4 cells in the compiler communications region (COMMON) so that other phases can determine which options were chosen.

The parameters processed by phase 01 include:

- From the system OPTION card: DECK, ERRS, LINK, LIST, LISTX, LVL, SYM, and XREF

- From the CBL card: CATALR, CLIST, COUNT, CSYNTAX, FLAGW, FLOW, LIB, LVL, OPT, QUOTE, SEQ, STATE, STXIT, SUPMAP, SXREF, SYMDMP, SYNTAX, TRUNC, VERB, VERBREF, VERBSUM, and ZWB
- From the LST option card: COPYPCH, DECK, LSTCOMP, LSTONLY, and PROC

If SYNTAX was specified, phase 01 turns off the bits for conflicting options. From the CBL card parameter BUF, phase 01 obtains the buffer size for the work files (see "Buffer Size Determination" below). From the CBL card parameter SPACE, phase 01 sets the print control character in the first byte of the buffer area for SYSLST.

In addition to the above parameters, phase 01 uses the system communications region to determine the date of compilation and the number of lines to be printed per page. The number of lines is passed to phase 00 as described in this chapter under "Information Returned to Phase 00." Phase 10 sets the RPTWR bit in the PHZSW1 cell.

#### BUFFER SIZE DETERMINATION

The compiler uses six buffer areas. Figure 10 shows which files use which buffers in each phase. Because buffers 1 through 5 are always used for work files, they are all the same size so that they can be used for different files from phase to phase. Buffer 6 is also used for a work file in phases 20, 21, 22, 50, and 51; therefore, it must be at least as large as the other buffers. Buffer 6 may have to be larger than the other buffers, however, because in phases 10, 11, 12, and 60 it is used for up to three double-buffered compiler files.

The buffer sizes of the compiler files are:

SYSLST	--	133 bytes
SYSPCH	--	81 bytes
SYSIPT	--	80 bytes
SYSLNK	--	82 bytes

Files SYS001 through SYS005 are assigned five buffers and may also use buffer 6 when it is available. The minimum buffer size for these files is 512 bytes each. This size can be changed by means of the BUF parameter on the CBL card. The maximum is 32,767 bytes. Specifying a larger size will cause a default to this value.

Phase	SYS001	SYS002	SYS003	SYS004	SYS005	SYSIPT	SYSLST	SYSPCH	SYSLNK
01			1,2	4,5		6,6	6,6 <sup>6</sup>		
05		3,3		4,5 <sup>9</sup>		6,6			
06		3,3		4,5					
07							6,6		
08		3,3		4,5			6,6	6,6	
10			1,2	5,5 <sup>8</sup>		6,6	6,6 <sup>6</sup>		
11		3,4		5,5 <sup>8</sup>		6,6	6,6 <sup>6</sup>		
12		3,4	1,2	5,5 <sup>8</sup>		6,6	6,6		
20		3,4	1,2	5,6					
22		3,4	1,2	5,6					
21		3,4	1,2	5,6					
25				5,6	1,2 <sup>4</sup>				
30		3,4	5,5	1 <sup>1</sup>			6,6		
40	3,4		2,5	1 <sup>7</sup>					
50	3,4	2,6	1						
51	3,4	2,6	5	1					
60	3,4	2 <sup>3</sup>	2,5	1			6,6	6,6	6,6
	3 <sup>2</sup>		5 <sup>3</sup>						
62	3,4		2,5						
	3,3 <sup>2</sup>								
63	3,4	2,5							
	3,3 <sup>2</sup>								
64	1	2,3	4	5					
65		4		1	1,2 <sup>4</sup>			6,6	6,6
61	1 <sup>5</sup> , 2 <sup>5</sup>			3 <sup>5</sup> , 4			6		
	3,3			1					
70			2 <sup>5</sup> , 5	1 <sup>7</sup>			6,6		
			5						

Note: The SYSSLB buffers are resident in phase 01.  
<sup>1</sup>SXREF or XREF only.  
<sup>2</sup>File 1 is single buffered for segmentation.  
<sup>3</sup>SYMDMP only.  
<sup>4</sup>SYMPMP requires a buffer of 512 bytes, which is the combined size of buffers 1 and 2.  
<sup>5</sup>OPT.  
<sup>6</sup>When LVL is specified, the SYS006 workfile replaces SYSLST for input to phase 80. Phase 80 output is in turn directed to SYSLST.  
<sup>7</sup>Used if SYNTAX is on.  
<sup>8</sup>Used only if LIB or LSTCOMP.  
<sup>9</sup>Used only if LIB.

Figure 10. Buffer Assignments

The minimum buffer space required for the compiler (that is, the total space required for buffers 1 through 6) is 2986 bytes. Buffers 1 through 5 require 2560 bytes (5 x 512 bytes). Buffer 6 requires a minimum of 426 bytes, as follows:

SYSLST:	133 x 2 =	266
SYSIPT:	80 X 2 =	160
Total		= 426

If the DECK and LINK options are specified, 3152 bytes are required for buffers as follows:

Buffers 1-5:		2560
Buffer 6:		
2 x SYSLST =		266
2 x SYSPCH =		162
2 x SYSLNK =		164
Total	=	3152

SYSPCH and SYSLNK reuse the space originally occupied by the SYSIPT buffers.

In a 60K byte partition, about 4K bytes are available for buffers. SYSLST block size (BLKSIZE) should not be larger than 605. For example:

Buffers 1-5:		2560
Buffer 6:		
2 x SYSLST =		1210
2 x SYSPCH =		162
2 x SYSLNK =		164
Total	=	4096

Phase 01 records the buffer sizes in phase 00 buffer control blocks. Phase 00 uses the control blocks to locate the current buffer for a file and to keep track of how much of the buffer has been used.

OPENING FILES

Phase 01 opens the four utility files (SYS001 through SYS004), SYSIPT, and SYSLST and, if required, SYSLNK, SYSPCH, SYSSLB, SYS005, and SYS006. It determines whether the files are operable and prints a message on SYSLST if a file cannot be opened.

INFORMATION RETURNED TO PHASE 00

Phase 01 passes the following information back to phase 00 in the manner indicated:

1. The flags indicating the compilation options via the PHZSW, PHZSW1, PHZSW2, and PHZSW3 cells in COMMON.

2. The LINES indication, by filling in an area in phase 00 called LINECNT, the address of which was passed to phase 01 by phase 00.
3. The beginning and ending addresses of the main free area (initial table area) by filling in an area, as in 2 above, called TAMAREA.
4. The address of the buffer area and the buffer lengths, by filling in an area, as in 2 above, called BUFCNLS.
5. The alternate name for the symbolic debug file, if requested on the CBL card (disk file only).
6. The date and time of the compilation via the DATE/TIME cells in COMMON.
7. The value of PMAP relocation factor from CBL card in PMAPADR cell in COMMON.

ERROR CONDITIONS

If any of the error conditions listed below are discovered by phase 01, an appropriate error message is placed into an internal phase 01 table by the QUE routine. Routine PRINT writes the messages out on SYSLST (SYS006 for LVL option) before control is finally returned to phase 00. Where necessary, an end-of-job indication is sent to phase 00 via the XY code (see Figure 5 in the chapter on phase 00), and phase 00 terminates the compilation. The specific error conditions are:

1. A file cannot be opened. Compilation is terminated only if the file is required.
2. The BUF parameter is invalid or insufficient. An alternate value is chosen or compilation is abandoned.
3. The virtual partition size is too small to accommodate the compiler. Compilation is terminated.
4. A premature end-of-file condition has been encountered on SYSIPT. Compilation is terminated.

LIB OPTION PROCESSING

If the LIB option has been specified, phase 01 scans the source program for BASIS and COPY statements. It performs syntax

analysis on these statements and copies library information with the remainder of the source program on SYS004 for subsequent phase analysis. The phase also

- Writes the source program listing on SYSLST (SYS006 for LVL option) if LIST is in effect and no Lister options are in effect.
- Generates internal sequence numbers for source program.
- Performs sequence error checking if SEQ is in effect.

If BASIS or COPY statements are syntactically incorrect, phase 01 writes error text on SYS003.

FEDERAL INFORMATION PROCESSING STANDARD  
(FIPS) FLAGGING

When the LVL option is specified, phase 01 enters the level character into byte 12 of the System Communication Region to indicate to phase 80 what level of the FIPS standard is to be flagged (A = low; B = low intermediate; C = intermediate; D = full standard). SYS006 receives the source listing that is used for input to phase 80 for FIPS flagging).



PHASE 05

Control is given to phase 05 (ILACBL05) only when the Lister option (LST) 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.

Input

The input to phase 05 is the COBOL source program. If NOLIB is in effect, input is read from the card reader. If LIB is in effect, input is read from SYS004 (output from the COPY preprocessor).

Output

The output from phase 05 is written on SYS002. 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 option listing.

## ERROR CONDITIONS

Phase 05 will recover from any syntax error that it detects in the COBOL source program. When such an error is detected, phase 05 inserts an error marker to identify those elements of the source program that are in error. A recovery marker is inserted at the point where the recovery takes effect. These markers are used by phase 08 to flag the incorrect source statements in the listing.

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 unusual termination of phase 05 occurs, the file written on SYS002 will be incomplete.

## PHASE 06

Phase 06 (ILACBL06), 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, pointers to the places where references to that item 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 SYS002 by phase 05. Input for subsequent passes of phase 06 is the output of the previous pass. That is, input will be read alternately from SYS002 and SYS004 (beginning with SYS002).

### Output

The output of phase 06 is written alternately on SYS004 and SYS002. Output of the first pass of phase 06 will always be written on SYS004 and output of the last pass will always be written on SYS002. The output file consists of the source program with cross-reference information embedded in it; the contents of the file will be formatted and printed by phase 08.

PHASE 07

The function of phase 07 (ILACBL07) is to print the first page (preface) of the Lister option listing.

Input

Phase 07 uses no input.

Output

The output of phase 07 is printed on SYSLST. It 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

## PHASE 08

The functions of phase 08 (ILACBL08) are as follows:

- Print the body of the Lister option listing
- Depending on the options specified for LST
  - Punch the reformatted source program on SYSPCH
  - Pass the reformatted source program, via SYS004 to phase 10 for compilation

### Input

Phase 08 reads input from SYS002. Input consists of the source program with embedded cross-reference information from phase 06.

### Output

The output of phase 08 consists of:

- The Lister option listing
- An internal card-image CCBOL source program
- A reformatted source deck.

The Lister option listing is printed on SYSLST. The internal card-image source

program, which may serve as input for subsequent compilation, is produced on SYS004 if the LSTCOMP option is in effect. The reformatted source deck is produced on SYSPCH if the Lister DECK 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 SYSLST and then deletes the page from storage. The process is repeated until all data from SYS002 has been processed. To produce the summary listing, phase 08 repositions SYS002 to the first record 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 not all of the footnotes will be printed in the Procedure Division of the Lister option listing.

PHASE 10

Phase 10 (ILACBL10) reads the source statements for the Identification, Environment, and Data Divisions, except the Report Section. As it reads the card images, it performs the following major functions:

- Encoding in Data IC-text (see "Section 5. Data Areas"), and storing in tables and in cells of the compiler communications area (COMMON) information from the Identification, Environment, and Data Division source statements.
- Analyzing the syntax of the statements read.
- Writing, on SYSLST (SYS006 for LVL option), the source program listing of the Identification, Environment, and Data Divisions (except the Report Section) if the LIST and NOLIB options are in effect.

Phase 10 includes several major working routines and the division processing routines.

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 GETWD, GETCRD, and GETDLM routines. These routines are also used by phases 11 and 12.

GETCRD Routine

The GETCRD routine reads the next card from SYSIPT, or from SYS004 if LIB or LST is in effect, stores its image into a work area called COMWRK, and writes a line on SYSLST (SYS006 for LVL option) if the LIST option was specified and no Lister options were specified.

GETWD Routine

The main functions of the GETWD routine are:

- Getting a logical unit from the input card image provided by the GETCRD routine, identifying and encoding it, and sending it to the calling routine for processing. A logical unit is defined as all the characters between one blank and the next.
- Generating a card number for each input card, starting with 1. The current card number is kept in a halfword labeled CURGCN.
- Making sure 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 BCD name, etc.).

GETDLM Routine

The GETDLM routine acts as the coordinator for the processing of the Identification, Environment, and Data Divisions. The major functions of the routine are:

- Looking 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 header, the end of the Data Division, or an end-of-file condition on the input device.

IDENTIFICATION DIVISION

The Identification Division scan routine (IDDSN) is entered immediately after the phase 10 initialization routines. The input is scanned for an Identification Division header. When one is encountered, the cell for the next logical unit is filled by GETWD and checked to see whether it is PROGRAM-ID. If it is, the program name is saved in the PROGID field 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 the segments in a segmented program.

After the PROGRAM-ID has been saved, the Identification Division is written on SYSLST (SYS006 for LVL option) if the LIST option was specified or on SYS006 if LVL was specified. If a DATE-COMPILED paragraph is included in the Identification Division, the information in the paragraph is deleted and the current date is inserted from COMMON.

#### ENVIRONMENT DIVISION

When the Environment Division header is encountered, the Environment scan routine (ENVSCN) searches for the Configuration and Input-Output Sections and, as each is found, branches to the routines that process it. These routines produce the file definition portion of Data-IC text (for text formats, see Data Areas section), which is 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 is processed in phase 10.

#### OBJECT-COMPUTER Paragraph

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 was set to 49 by phase 00.

#### SPECIAL-NAMES Paragraph

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

DECIMAL-POINT Clause: The KDECML field is changed from "period" to "comma". Thereafter, when phases 10, 11, and 12 scan

numeric and floating-point literals, commas instead of periods are recognized as decimal points.

UPSI-n Clause: Dummy dictionary entries are created from the data in each UPSI-n clause and are entered in the UPSTBL table. The dummy dictionary entries are so constructed that phase 22 can enter into the dictionary an LD entry from each UPSI-n and each mnemonic-name, and a condition-name entry from each condition-name-1.

Function-name IS Mnemonic-name Clause: An entry is made in the SPNTBL table for each mnemonic-name. These table entries are used by phase 11 during processing of the Procedure Division. When phase 11 scans ACCEPT or DISPLAY statements, it replaces the mnemonic-name with the proper console or device name by checking this table. When phase 11 scans a WRITE...AFTER ADVANCING statement, it replaces the mnemonic-name with the proper carriage control word.

#### INPUT-OUTPUT SECTION

The routines that process the Input-Output Section build and use the ENVTBL and QNMTBL tables. The QNMTBL table contains variable-length names; the ENVTBL table contains pointers to each entry in the QNMTBL table. These tables are released later in phase 10. Their formats are given in "Section 5. Data Areas".

#### File-Control Paragraph

The SELSCN routine produces one partial Data IC-text entry for each SELECT sentence. These entries contain only file information and are stored in the ENVTBL table. Later, during Data Division processing (see "File Section" in this chapter), these entries are used to produce complete FD entries in Data IC-text.

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

#### VSAM File Processing

For FILE STATUS clause processing, SELSCN passes control to the file status

routine to enter the FILE STATUS dataname into the QNM\_TBL and to set a pointer to QNM\_TBL in the corresponding ENV\_TBL table field. A corresponding bit is also turned on in the ENV\_TBL 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 QNM\_TBL and to set a pointer to QNM\_TBL in the corresponding ENV\_TBL table field. A corresponding bit is also turned on in the ENV\_TBL to indicate that a PASSWORD clause has been specified.

### I-O-Control Paragraph

When the I-O-Control paragraph header is encountered, the ENVSCN routine calls the pertinent routines for processing the SAME, RERUN, MULTIPLE FILE TAPE, and APPLY clauses.

SAME Clause: For each clause encountered, the files named are entered into one of the following tables: for SAME AREA, the SATBL table; for SAME RECORD AREA, the SRATBL table; for SAME SORT AREA, the SSATBL table.

At the end of the 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 the number 2; and so forth. Similarly, the first SAME RECORD AREA clause is assigned the number 1; the second SAME RECORD AREA clause is assigned the number 2; and so forth. The same procedure is followed for SAME SORT AREA clauses.

The ENV\_TBL table is then searched for all the files named in the SAME clauses, and appropriate numbers are inserted into these entries of the ENV\_TBL table to identify the SAME clauses in which the files were named. For example, if three SAME RECORD clauses were specified, each file named in the first clause would have "1" in the SAME RECORD field of its ENV\_TBL table entry; each file named in the second clause would have "2" in the SAME RECORD field; each file named in the third clause would have "3" in the SAME RECORD field.

The appropriate switches are set in the ENV\_TBL table entries. At this time, the SATBL, SRATBL, and SSATBL tables are released.

RERUN Clause: Format 1: An entry for the file name is made in the CKPTBL table. In

the FNV\_TBL table entry for this file, the CKPTBL bit is set to 1 and a pointer to the CKPTBL table entry is inserted. The RERUN bit in the PIOTBL table entry associated with the file-name is later set to 1 during Data Division processing. The RERUN switch in COMMON is turned on.

Format 2 (SORT-RERUN): An entry is made in the CKPTBL table, with the INTEGER field set to zero.

MULTIPLE FILE TAPE Clause: The MULTIPLE FILE TAPE switch is set to 1 in the ENV\_TBL table entry for the file named in the clause. A number indicating the file's position on the tape is placed in the POSITION INTEGER field.

APPLY Clause: For each option, a switch is set in the ENV\_TBL table entry for the file named in the clause.

Option 1: The WRITE-ONLY switch is set.

Option 2: The EXTENDED-SEARCH bit is set.

Option 3: The WRITE-VERIFY switch is set.

Option 4: The CYL-OVERFLOW switch is set. The number of tracks is converted to binary and placed into the CYL-OVERFLOW-TRACKS field.

Option 5 (Format 1): The MASTER-INDEX switch or the CYLINDER-INDEX switch is set. The device-number is stored in the DEVICE-TYPE-CODE field.

Option 5 (Format 2): The CORE-INDEX switch is set. The data-name is stored in the QNM\_TBL table, and a pointer to it is entered in the ENV\_TBL table entry for this file.

### DATA DIVISION

When the Data Division header is encountered, the GETDLM routine calls the DDSCN routine, which in turn calls the routines that process the File, Working-Storage, and Linkage 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:

- File and record information is entered into a work area (ICTEXT).

- Entries are made in the OD2TBL, QNMTBL, FNTBL, and RCDTBL tables.
- Information for FDs in the ICTEXT work area is merged with the corresponding ENVTBL table entry.
- Data IC-text for FDs, LDs, and SDs is generated and written on file SYS003. 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.

The routines that process the Data Division use the ENVTBL and QNMTBL tables (additional entries are made in the QNMTBL table) and build the OD2TBL, FNTBL, and RCDTBL tables (see "Section 5. Data Areas" for formats). All of these tables except the ENVTBL and QNMTBL tables are passed to phase 11. The PIOTBL table is also used by phase 21, and the OD2TBL table by phase 22 and phase 25. The PIOTBL table entries are filled in by phase 11 during Procedure Division processing and indicate which OPEN options and input/output verbs are used for each file; the OD2TBL table is used to generate Q-Routines (object module subroutines used to calculate variable lengths for OCCURS...DEPENDING ON fields and variably located fields following the variable-length fields). The OD2TBL table is also used in building the debug file if SYMDMP has been specified (see "Building the OBODOTAB Table" in the chapter "Phase 25").

## FILE SECTION

After the File Section header is encountered, the DDSCN routine calls the appropriate routines to process FDs, SDs, and LD's in the source program.

### File Description Entries

Each File Description entry (FD) is analyzed, and information from the clauses is entered into the ICTEXT work area. This information, which includes the file-name and the LABEL RECORDS switches, is merged with some of the ENVTBL information and placed into the FNTBL table. This ENVTBL information includes the ACCESS RANDOM and mass-storage switches and the CKPTBL bit that were set during Environment Division processing. A PIOTBL table entry is set up, and pointers to this entry are placed

in the FNTBL and ENVTBL tables. The PIOTBL table contains binary zeros at this time. Variable-length names such as LABEL RECORD data-names are entered into the QNMTBL table, and pointers to these entries are placed in the work area.

Note that the REPORT clause in the FD statement requires special processing; this is described in the phase 12 chapter.

When this processing has been completed, the ENVTBL entry and the file description information from the work area are merged into Data IC-text FD entries. Names from the QNMTBL table are located (from their ENVTBL table 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 entry (SD) is placed in work area ICTEXT and is used to generate an SD entry of Data-IC text.

### Record Description Entries

When a level-number entry (LD) is encountered, the DDSCN routine calls a routine to analyze the entry and store information from the clauses into work area LDTEXT. (This area is the same physically as ICTEXT.) If the OCCURS clause with the DEPENDING ON option is included, the object of the clause and its qualifiers are entered into the OD2TBL table (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. LD-text is an internal phase 10 text.)

For 01-level items, an RCDTBL table entry is made, consisting of a pointer to the most recent file-name entry in the FNTBL table, followed by the record name. As a result, each RCDTBL table entry contains a pointer to a corresponding FNTBL table entry, which in turn contains a pointer to a corresponding PIOTBL table entry (see "Section 5. Data Areas"). This relationship is used by phase 11 when processing WRITE and REWRITE statements to relate records to their associated files.

When all the information about the data-name has been stored in LDTEXT, the contents are written out as an LD entry in Data IC-text on SYS004.



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 table entries are made.

SYNTAX ANALYSIS

Phase 10 performs a syntax analysis of the Identification, Environment, and Data Divisions during division processing. Included are such things as checking for division headers and making sure that the PROGRAM-ID clause appears in the first paragraph of the Identification Division. If user errors are detected during syntax analysis, E-text is generated.

## PHASE 12

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 SYSLST (SYS006 for LVL option), if the LIST and NOLIB options are in effect and no Lister options are in effect.
- Records information for later phases in TAMER tables and in COMMON cells.

Phase 12 reads its input from SYSIPT or from SYS004 if the LIB or IST option is in effect. It writes its output, the RWS, in the form of Data IC-text on SYS003 and in P0-text on SYS002. Any E-text produced is also written on SYS002, intermingled with the P0-text. The input and output are summarized in Figure 11. The RWS is described in Appendix C.

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 RWRTBL table (first REPORT

clause only), sets a flag bit in the P1BTBL table (first REPORT clause only), and enters the report name into the RWRTBL 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 RPTWR bit in the PHZSW switch 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 11:

- 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.

### PROCEDURE DIVISION VERBS

Upon encountering an INITIATE, GENERATE, or TERMINATE verb, phase 11 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.

## 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.

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 01-level 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 01-level 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 01-level statement.

The PROC01-FLUSH or PROC01-PROC02-FLUSH loops continue until phase 12 has generated

RWS group routines for all of the 01-level 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

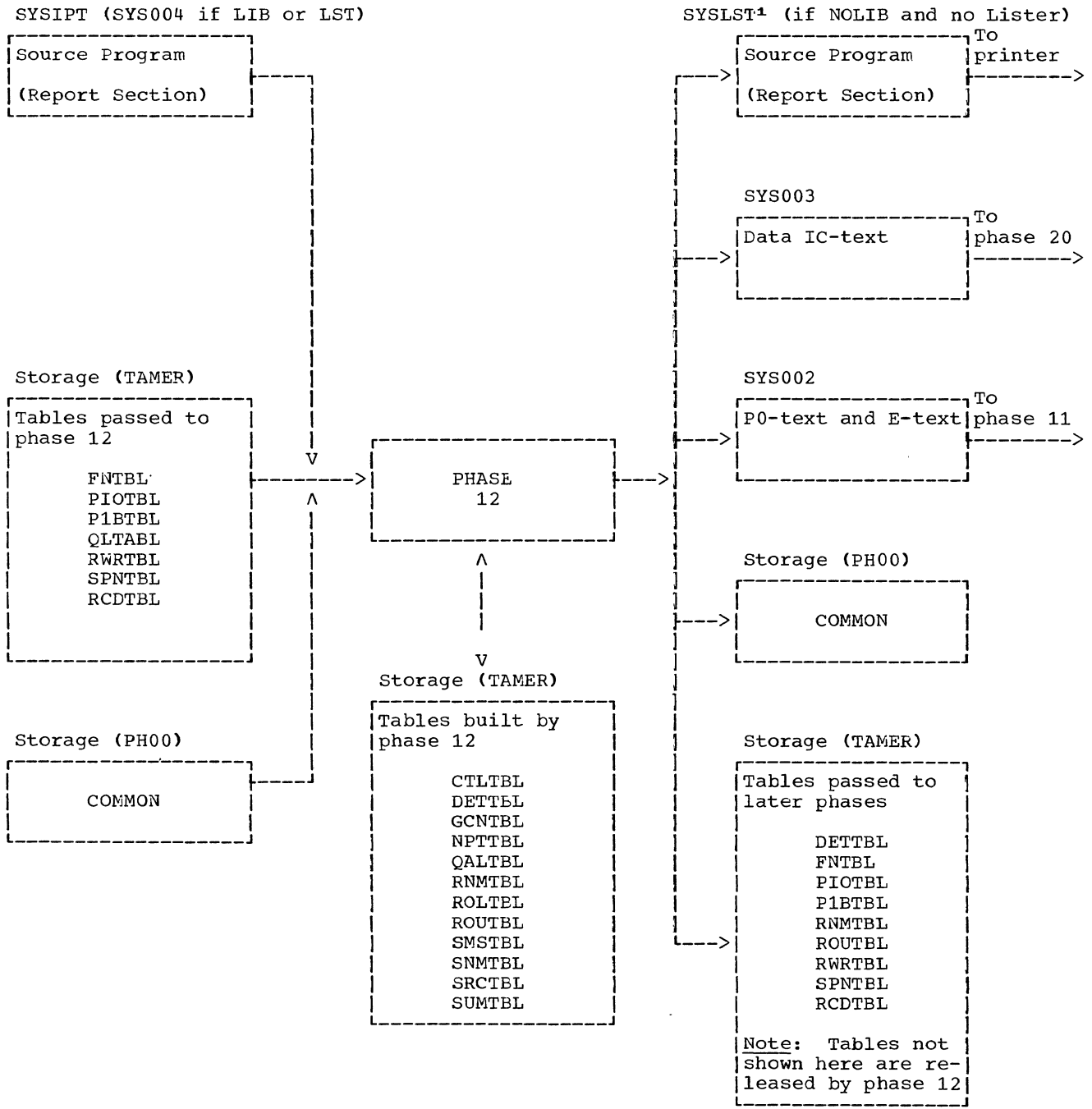
The RDSCAN routine is responsible for processing the RD entry of the source program. After first ensuring that an RWS should be generated (by determining whether the RWRTBL table is primed), it reads each logical unit of the RD and processes it. The routine operates in a loop-type scan, checking each 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.

## ROUTINE PROC01

Routine PROC01 processes the 01-level 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 01-level 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 routine. Since the compiler has, at that point, enough input to begin the group routine, the first part of that routine is generated here.



<sup>1</sup>SYS006 is used if the LVL option is in effect.

Figure 11. Phase 12 Input/Output Flow

ROUTINE PROC02

Routine PROC02 is entered when routine GETDLM encounters an 02-49-level entry or, at entry point PRO2A, during PROC01's scan of an 01-level 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 onto SYS002 (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 SYS003 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 (or 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 SYS002.

GENERATING THE SOURCE LISTING

As each record is read from SYSIPT, a check is made to determine if the LIST option is in effect. If so, the source statement is copied out onto SYSLST (or SYS006 for LVL option).

INFORMATION FOR LATER PHASES

During its processing, phase 12 stores various types of information for later phases to use. For example, phase 12 builds the ROUTBL table, which contains the specific GN numbers assigned to certain RWS routines. Phase 11 needs this information in order to process INITIATE, TERMINATE, GENERATE, and USE BEFORE REPORTING statements. Such items are stored in TAMER tables and in cells in COMMON. For more details on this subject, refer to Figure 11 and to "Communications Region" and "Table Formats" in "Section 5: Data Areas."

## PHASE 11

Phase 11 (ILACBL11) reads the source statements of the Procedure Division. It is entered via phase 00 when phase 12 encounters the Procedure Division header, or when the GETDLM routine in phase 10 encounters an end-of-file condition. As it reads each card in the Procedure Division, phase 11 does the following:

- Encodes the Procedure Division into P0-text.
- Enters procedure-names into the dictionary.
- Writes the Procedure Division on SYSLST (SYS006 for LVL option), if the LIST and NOLIB options are in effect and no Lister options are in effect.
- Generates error text (E-text) for syntax errors it encounters.

The phase 11 routines first process the out-of-line procedures contained in the Declarative Sections. (Processing declaratives is described later in this chapter.) Then the in-line program is processed.

Tables passed from phase 10 and used by phase 11 include the PIOTBL, FNTBL, RCDTBL, P1BTBL, and SPNTBL tables. Tables passed from phase 12 and used by phase 11 are the DETTBL, RNMTBL, ROUTBL, and RWRTBL tables. The PNTABL and PNQTBL tables are built during phase 11. If the VERBREF or VERBSUM option is in effect, phase 11 will create the VERBDEF Tamer table. Formats for tables, text entries, and dictionary entries are given in "Section 5. Data Areas."

Phase 11 functions are performed under the control of the PDSCN routine and the two major working routines GETCRD and GETWD (which supply input to PDSCN). The GETCRD and GETWD routines are described under "Major Working Routines" in the phase 10 chapter. These and all other routines used by both phases do not remain in storage from phase 10, but are reloaded with phase 11.

If the VERB, VERBREF, or VERBSUM 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.

## ENCODING THE PROCEDURE DIVISION

A major activity of phase 11 is writing P0-text. This text is, roughly, the source program Procedure Division encoded into a form acceptable to later phases. Logical units (source program character configurations) 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."

## PROCESSING 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 into the dictionary and written in P0-text. Some dictionary attribute bits are set when the entry is made, and others are moved in from the PNTABL or PNQTBL tables later when the attributes are known. The building of the PNTABL and PNQTBL tables is described below under "Processing Verbs." Entering PNTABL and PNQTBL information into the dictionary is described later in this chapter.

## Priority Checking for Segmentation

For each section-name, the segmentation priority is entered into 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  
(this cell either was set by phase 10 when  
processing the SEGMENT-LIMIT clause or  
contains a default segment-limit of 49).  
If the priority number of the section-name  
is less than SEGLMT, it means that the  
section is part of the root segment. In





this case, the priority number of the section-name is entered into the dictionary as zero. If the priority number exceeds SEGLMT, the specified number is entered into the dictionary.

Each time a section-name is found whose priority exceeds the value of SEGLMT, a phase 11 switch 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 'FP'. If the switch is on, SEGLMT is left as it was. The value of SEGLMT is used by later phases to determine whether or not the program is segmented.

#### PROCESSING VERBS

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

#### Procedure Branching Verbs

When the procedure branching verbs (ALTER, EXIT, GO TO, and PERFORM) are processed, information about how a procedure-name is used is entered into the PNTABL or PNQTL table. If the procedure-name to be entered is qualified by a section-name, the procedure-name and its qualifier are entered into the PNQTL. If the procedure-name is not qualified, it is entered into the PNTABL. Entries from the tables are used to set some of the attribute bits in the dictionary entries for the procedure-names. A unique bit is turned on in the attributes field for the procedure-name according to their use.

GO TO: The left-hand name (the procedure-name appearing in Area A, not the object of the GO TO) is entered into the PNTABL table. If the DEPENDING ON option is used, no entry is made.

EXIT: The left-hand name is entered into the PNTABL table.

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

PERFORM...THRU: The procedure-name following the THRU is entered into the table. If the THRU option is not used, the procedure-name following PERFORM is entered.

#### Input/Output Verbs

Switches are set in the PIOTBL table entry for the file named in input/output verbs. These switches tell phase 21 how the file was used. In addition, the following processing takes place. The REDSVB routine also checks for the word NEXT in the READ verb. If it is present, the routine turns on the appropriate bit in the PIOTBL table.

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

DELETE: The FILSVB subroutine checks that the filename 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 file-name is valid.

WRITE, REWRITE: The record name is sought in the RCDTBL table; the RCDTBL entry contains a pointer to the FNTBL table, which is used to find the file-name. The file-name is then included in the P0-text entry (in the form "WRITE file-name record-name"). If the ADVANCING option of the WRITE statement is used with the mnemonic-name option, it is sought in the SPNTBL table and replaced with the proper function-name.

#### Other Verbs

ACCEPT, DISPLAY: If a mnemonic name is used, it is sought in the SPNTBL table and replaced by the proper function-name.

DEBUG: The attribute bits in the dictionary entry for the paragraph referred to are set.

EXHIBIT: A special EXHIBIT data-name is generated.

READY: The TRACE switch in COMMON is set.

SORT/MERGE: If the USING or GIVING option is specified, the appropriate bits are set in the PIOTBL table entries for the files named. If these files are also specified in label or error processing declaratives, the GNs for the declaratives are appended to the file-name.

Report Writer Verbs: See "Appendix C:  
Report Writer Subprogram."

#### PROCESSING DECLARATIVES

When a Declaratives Section is encountered, the section-name (and any paragraph-names as they are encountered) are entered into the dictionary. A PNTABL entry is made for the section-name; the declarative bit and the bit identifying the type of declarative are set. Every paragraph-name in the section is also entered into 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 ON file-name, the GNs are entered into the appropriate fields of the PNTABL entry for the file. If the USE sentence specified ON INPUT, ON OUTPUT, or ON I-O, the GN number is entered into a particular OPEN option work area.

For each file-name specified in an OPEN statement, the corresponding PNTABL table entry is inspected. If GNs were entered into the PNTABL during declarative processing, they are inserted into the P0-text. Otherwise, the work areas for the particular OPEN option are searched for the appropriate GNs for this file. For an error declarative, if a GN for ON INPUT is found in the work area, the GN number is inserted into all OPEN INPUT P0-text entries. The GNs for label declaratives are appended only for files whose PD entries 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 GVPNTABL table when a VSAM file has been specified in the ON option of the declarative.

#### ENTERING PROCEDURE-NAMES IN THE DICTIONARY

A dictionary entry is made for all procedure-names (PNs) upon their point of definition (appearance in Area A) in the source program. The entry's 2-byte

characteristics field is later filled in from the flag field of either the PNQTL or PNTABL table if the PN (1) follows the PERFORM in a PERFORM...THROUGH statement, (2) follows the PERFORM in a PERFORM statement without the THROUGH, (3) is altered by an ALTER statement, (4) follows TO PROCFFD TO in an ALTER statement, (5) precedes a GO TO or an EXIT statement, (6) is referred to or defined in a DEBUG paragraph, (7) is a dummy name, or (8) appears in the Declaratives Section. The three fields are identical.

Each time such a PN is encountered, the PNQTL table (if it is a qualified name) or the PNTABL table (if it is not) is searched to find an earlier entry for it. If one is found, the flag bytes are modified; if not, a complete entry is made. At the end of the Procedure Division, or the section currently being processed if the Procedure Division is divided into sections, a search is made in the dictionary. When an entry is found for that PN, the characteristics field is modified, and the PNTABL or PNQTL entry is deleted. A new table entry is made for that PN if it is encountered in a later section.

Whenever the Procedure Division is divided into sections, the technique used to search the dictionary is affected. Phase 11 calls the appropriate TAMER ACCESS routine, giving it the name of a particular Procedure Division section. All PNs entered in the dictionary from that section are then searched. If the search is for a PN from the PNTABL table and the dictionary has been searched without finding the PN, a search bit is turned on in the table entry. The next time the dictionary is searched for this PN, only entries from the newly processed section will be searched. In the case of the PNQTL table, the table entry contains the section-name qualifier. Thus, if the section named in the table entry has been processed, the dictionary entry is found among the entries for that section. If the section-name has not yet been encountered, no search is made, and no search bit, therefore, is needed.

Figure 12 shows an example of such a search. Note that the figure does not show the entry formats. Dictionary formats and table formats are shown in "Section 5. Data Areas."

#### DUMMY ENTRY FOR PHASE 80

Phase 11 places a dummy element after the source statements; this dummy element is used by phase 80.

DICTIONARY

Point of definition is in section	Procedure name
1	PN1 PN2
2	
3	PN5 PN6

PNTABL

Section of Table Entry	Procedure Name	Action Taken
	PN2	Dictionary entry found on first search. Table entry is deleted.
1	PN6	On first search, the dictionary entry not yet made, so search bit is set. On second search, only entries from section 2 are searched. Since the name is still not entered in the dictionary, the search bit remains on. On third search, only entries from section 3 are searched. Entry is found and the table entry deleted.
3	PN5	On the third search, the whole dictionary is searched for this name, which is in the table for the first time. Dictionary entry is found and the table entry is deleted.

PNQTBL

Section of Table Entry	Qualified Procedure Name	Action Taken
2	PN1 in	On the second search, the entries from section 1 are immediately searched. The entry is found and the table entry deleted.

Figure 12. Entering PNTABL and PNQTBL Information into the Dictionary

PHASE 20

Phase 20 is the third of five phases that process the Data Division. It follows phase 11, overlaying it in storage. After it is loaded, storage contains phase 00 (including COMMON), the TAMER table and dictionary area, and phase 20. 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 SYS004 (the format of the entry is called ATF-text) and reads the next Data IC-text element, continuing until SYS003 has been exhausted. All Data IC-text for FDs, SDs, and keys for table handling and any E-text encountered is copied unchanged onto SYS004.

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 13.

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 01-49, 66, 77, or 88. If so, BEGIN stores the current input card number into a halfword in COMMON called CURCRD 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.

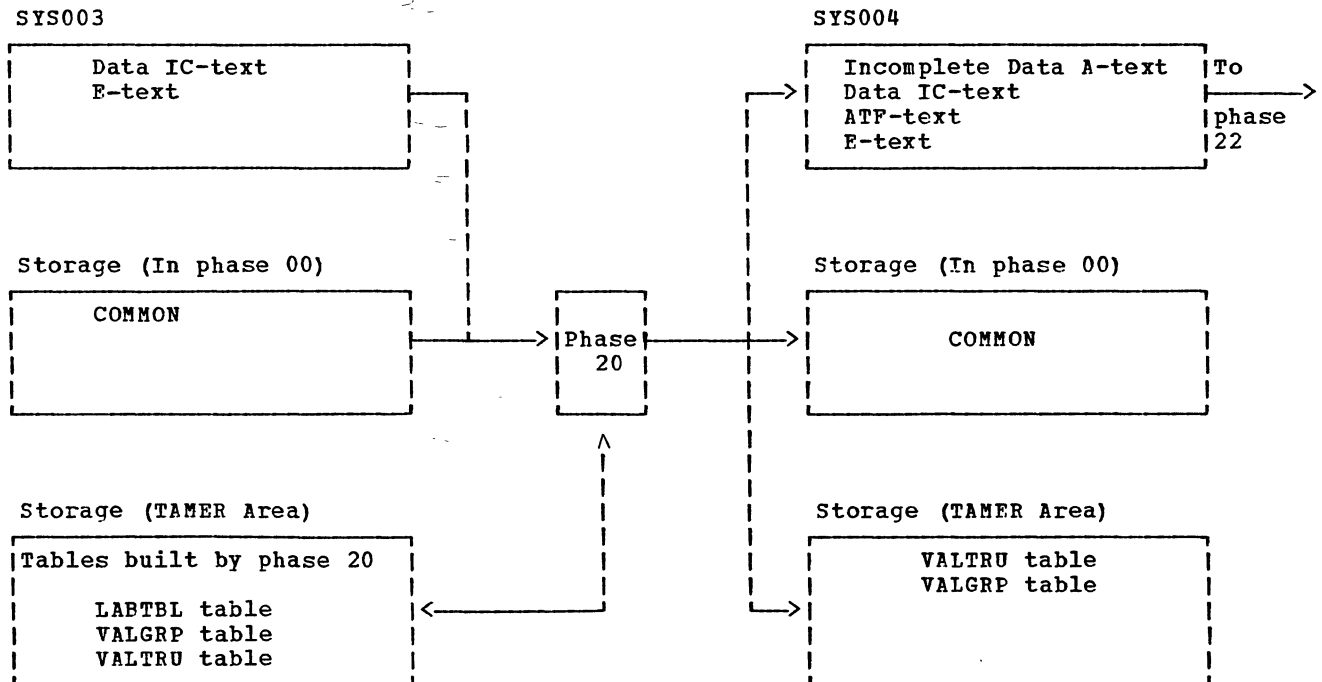


Figure 13. Phase 20 Input/Output Flow

## 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 MBBSRN. 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 30.

If there was a COPY flag in the Data IC-text, routine COPYRN picks up the name from the text so that the data-name of the entry can replace the COPY library-name in the copied entry.

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, 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 SYS003. 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. For a critical program break or EOF, routine FILEST returns control to routine BEGIN.

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 is the fourth of five phases that process the Data Division. (For an overview of that processing, refer to Figure 4 of the introduction to this book and to Diagram 1 in the Diagrams section.) Phase 22 follows phase 20, overlaying it in storage. Its major functions are:

- Producing dictionary entries.
- Completing Data-A text.
- 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 SYS004 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 SYS004 has been exhausted. All Data IC-text for FDS and SDs and any E-text encountered is copied unchanged onto SYS003.

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 14.

BUILDING DICTIONARY ENTRIES

Phase 22 stores the current card number for each input card in a halfword in COMMON labeled CURCRD. 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 LDTXT obtains a dictionary pointer for the item by calling an ACCESS routine, GETPTR. (ACCESS routines are described in "Appendix A. Table and Dictionary Handling.")

A delimiter pointer is needed for group items. Level-88 entries are put into the dictionary directly by the input routine READF4.

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 file attributes is determined by the access method specified; phase 22 determines that in making the skeleton dictionary entry. Routine FSTXT assigns the next DTF number to the file and writes the Data IC-text for the FD on SYS003. 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 SYS003 (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.

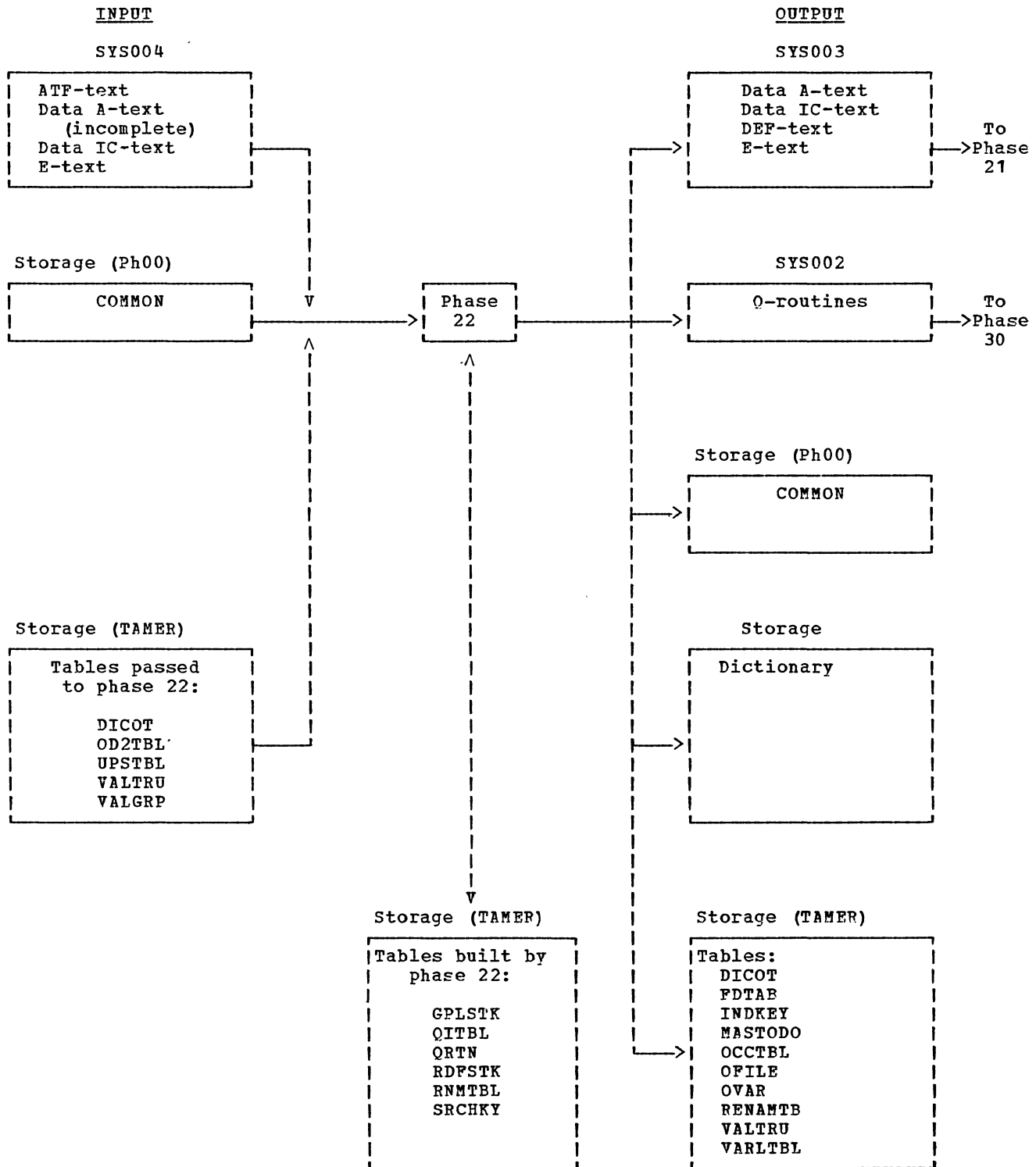


Figure 14. Phase 22 Input/Output Flow



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 SDTXX 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, 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 Report Section, and to each file (FD, RD, and SD entry).
BLL	BLLCTR	BLL numbers are assigned to the Linkage Section and to label records.
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 A-text element is written out, 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 77-level 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. 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.

Table RNMTBL is also used if there are a series of items with REDEFINES clauses.

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.

#### 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 second BLL. All level-77 items and all group items starting with level-01 in the Linkage Section are assigned unique BLL numbers.

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

DOS UPSI Feature Names: When routine INIT first receives control, it checks to see if phase 10 created the UPSTBL table. If so, it calls routine UPSI to enter the function-names, mnemonic-names, and condition-names into 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 60 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. The format of COMMCN, the tables, and the texts are described in "Section 5. Data Areas."

#### 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 and QVAR tables are passed on to phase 30; 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 table 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 30. 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 table GPLSTK 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 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 or not 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 (prefix 01 or 02) are entered into the SRCHKY table by routine READP4 while the dictionary is being built. This table is used for syntax checking whenever the names are encountered.

If CSYNTAX is specified and an error is detected, the syntax option bit in COMMON is forced on and conflicting options are forced off.

#### 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 main storage for phase 30. The VALTRU table is built by phase 20 and described under that heading.

The OVAR and QFILE tables are built during Q-Routine generation and stored for use by phase 30. They are discussed above under "Q-routine Generation."

During predictionary processing (described above), routine SRH200 creates the INDKEY table.

The FDTAB table is built for phase 21.

If the SYMDMP 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 OCCTBL table, the MASTODO table, the VARLTBL table, and the RFNAMTB table.

## PHASE 21

When phase 21 is loaded into storage, all data items except FDs and SDs have been translated from Data IC-text into Data A-text, and the dictionary is complete except that the FD and SD entries are dummy entries without data attributes written by phase 22. After phase initialization, routine BEGINPH is given control. This routine reads each record from SYS003 and determines the action to be taken. For an A-text or E-text element, the two-byte prefix attached in phase 20 or 22 is removed, and the element is copied onto SYS004; FD and SD elements are selected for processing by phase 21; all other records are copied unchanged onto SYS004.

From the FD and SD Data IC-text elements and from information stored by previous phases in the DICOT, PIOTBL, and PDTAB tables, phase 21:

- Completes the dictionary entries.
- Generates the required DTFs.
- Generates input/output areas (buffers).
- Writes the Data A-text for the DTFs and buffers onto SYS004 and PO-text for VCONS onto SYS002.

### COMPLETING DICTIONARY ENTRIES

When phase 21 encounters an SD record, routine SORTPROC reserves space for a buffer according to the maximum record size, generates a BL to point to the sort area, and fills in the SD dictionary entry. When the phase encounters an FD record, routine SETDIC fills in all fields of the dictionary which can be picked up directly from that record. Routine ACCMODE determines the access method specified and enters it into the dictionary.

Phase 21 builds the FD dictionary entries, the File Information Block (FIB), and the IND2TBL table (indexed file only) for VSAM files.

### GENERATING REQUIRED DTF'S

To develop a DTF, phase 21 checks the FD for clause compatibility, collects the parameters common to more than one type of

DTF, and selects the proper DTF generator. Each DTF generator then determines the number of DTFs required, reserves space for additional file information in the Pre-DTF, fills in certain fields of the Pre-DTF, and develops the body of the DTF itself.

### CLAUSE COMPATIBILITY

The compatibility testing performed by phase 21 is primarily a check to determine if the clauses specified are compatible with the file description. For example, an APPLY CORE INDEX is acceptable only for an ISAM file.

### COMMON PARAMETERS

Certain parameters are common to many or all types of DTFs and are determined before entry into the DTF generator. The remaining parameters are established by the particular DTF generator specified. Of the common parameters, the size and form of the records of a file are most important.

### Record Size

Routine RECCONT processes the RECORD CONTAINS clause. With only one exception the record size is the maximum size of any record described as a data record in the File Description. This value is determined in phase 22, and is passed to phase 21 in the PDTAB table. The one exception to using the maximum calculated size is the case where the clause RECORD CONTAINS N1 TO N2 CHARACTERS is specified, the value of N2 is less than the maximum calculated size, and there is more than one OCCURS clause with the DEPENDING ON option in the record description. In this case, the value of N2 is used as the maximum possible record size.

### Record Form

Prior to any attempt at determining the record form, the BLOCK CONTAINS clause is checked by routine BLKCTNS for basic

compatibility. If any errors are found, the clause is either dropped from further consideration or altered to an assumed acceptable format.

Every supportable DTF used in Full American National Standard COBOL requires an entry for its RECFORM parameter. The allowable entries are selected from the six possible kinds shown in Figure 15.

	CLASS= 'UR'	CLASS= 'UT'	CLASS= 'DA'		
		Tape and Card and Printer	Sequen- tial Disk	Direct Access Method	Indexed Sequen- tial Method
FIX UNB	Yes	Yes	Yes	Yes	Yes
FIX BLK	No	Yes	No	No	Yes
VAR UNB	Yes	Yes	No	No	No
VAR BLK	No	Yes	No	No	No
UNDEF	Yes	Yes	Yes	No	No
SPANNED	No	Yes	Yes	No	No

Figure 15. RECFORM Parameters Supported

The four factors which influence the selection of the record form are:

1. RECORDING MODE Clause. This clause has five possibilities: omitted, F, V, U or S.
2. RECORD CONTAINS Clause. This clause has three possibilities: omitted, N1 TO N2 CHARACTERS, or N2 CHARACTERS.
3. BLOCK CONTAINS Clause. This clause has five possibilities: omitted, N1 TO N2 RECORDS, N2 RECORDS, N2 CHARACTERS, or N1 TO N2 CHARACTERS.
4. Variability of the record descriptions found during phase 22 processing. This value is found in table FDTAB, referred to as VARIND and is either set or clear (FIXED or VARIABLE).

There are 150 possible combinations within these four factors. Most of the combinations are errors, but each is checked by routine CHKMODE, and a message is issued in E-text, if appropriate.

These four factors are encoded into a single byte which is used as an index (displacement) into a 156-byte error table, MODTAB. The referenced byte in the MODTAB table contains, in coded form, the assumed record form, an indicator to show if an error has occurred, and four bits to indicate the clause or clauses which are to be ignored.

After checking the table results, the recording mode can be determined, making assumptions where an error condition exists. If CHARACTERS or more than one record is specified in the BLOCK CONTAINS clause, the file is assumed to be blocked. If the block contains only one record, the file is assumed to be unblocked.

#### SELECTING THE DTF GENERATOR

After establishing the variables, phase 21 determines which DTF generator to use: GENDTFCD, GENDTFPR, GENDTFMT, GENDTFSD, GENDTFDA, GENDTFIS, GENDTFDU. The generator checks the OPEN options specified for the file to determine what type of DTF to generate. The OPEN options corresponding to each type of DTF are shown in Figure 16. Descriptions of the DTFs may be found in IBM DOS/VS Supervisor and I/O Macros, Order No. GC33-5373.

Each DTF generator makes entries in two tables, the BLTABL and the BUFTAB. The BLTABL table contains an entry for each FD in the program. It is used later in the phase, when buffers are generated, to determine how to initialize the base locator (or locators) for the file. The BUFTAB table contains an entry for each buffer area address constant which must be filled into a DTF. This table is also used when the buffers are generated. The formats of these tables are shown in "Section 5. Data Areas."

Phase 20 assigns a DTF number for each file. Every file having more than one DTF has secondary DTF numbers assigned consecutively. Counters located in COMMON are incremented each time a new number is assigned. At execution time, pointers in the TGT and PGT indicate the particular DTF for the file being processed. If all necessary DTFs have been generated for a file, routine ENTDIC enters the attributes of the file in the dictionary area reserved by Phase 22. When the next FD record is encountered, variables are again determined. This process continues until all the input has been processed.

DTF Type	File Type	Options
DTFCD	Card	Input
DTFCD	Card	Output
DTFPR	Printer	Output
DTFMT	Tape	Output
DTFMT	Tape	Input
DTFMT	Tape	Input, Reversed
DTFSD	Mass Storage	Sequential, Input
DTFSD	Mass Storage	Sequential, Output
DTFSD	Mass Storage	Sequential, I-O
DTFDA	Mass Storage	Direct, Sequential Access, Input
DTFDA	Mass Storage	Direct, Random Access, Input
DTFDA	Mass Storage	Direct, Random Access, Output
DTFDA	Mass Storage	Direct, Random Access, I-O
DTFIS Load Mode	Mass Storage	Indexed, Sequential Access, Output
DTFIS Retrieve Mode	Mass Storage	Indexed, Sequential Access, Input, I-O
DTFIS Retrieve Mode	Mass Storage	Indexed, Random Access, Input
DTFIS Retrieve Mode	Mass Storage	Indexed, Random Access, I-O <sup>1</sup>
DTFIS Add/ Retrieve Mode	Mass Storage	Indexed, Random Access, I-O <sup>2</sup>
DTFDO	Diskette Unit	Input Sequential, Output Sequential

<sup>1</sup>If there are no WRITE verbs for this file.  
<sup>2</sup>If there are WRITE verbs for this file.

Figure 16. OPEN Options and DTFs

DETERMINING THE NUMBER OF DTF'S

The number of DTFs is determined by the number and type of OPEN statements for each file. The DTF generator determines this from entries in the PIOTBL table. For a DTFCD or DTFPR file, only one DTF need be generated. For a DTFMT file, a maximum of three DTFs may be needed -- one for each INPUT, INPUT REVERSED, and OUTPUT. DTFSD files may also require three DTF's, one each for INPUT, OUTPUT, and I-O. Only one DTF is needed for a DTFDA file. It may be used for INPUT, OUTPUT, and I-O. Only one DTF is needed for any ISAM file description.

PRE-DTF AREA

Before phase 21 develops the body of a given DTF, it reserves space immediately preceding the DTF in storage for PNS, pointers, and file description information. In some cases, phase 21 fills in certain fields of this space. The area is known as the Pre-DTF.

DTFMT and DTFSD Pre-DTFs

For magnetic tape and sequential disk files, a 24-byte Pre-DTF is reserved in front of the DTF. The contents of this Pre-DTF are shown in Figure 17.

DTFDA, Random Access, Pre-DTF

For a file whose organization is direct and which will be accessed randomly, a variable-length Pre-DTF is reserved in front of the DTF. Figure 18 shows the contents of the Pre-DTF for a file with absolute addressing; Figure 19 shows the contents for a file with relative addressing.

DTFDA, Sequential Access, Pre-DTF

For a file whose organization is direct and which will be accessed sequentially, a 31-byte Pre-DTF is reserved in front of the DTF. Figure 20 shows the contents of the Pre-DTF for a file with absolute addressing; Figure 21 shows the contents for a file with relative addressing.

Location	Length (Bytes)	Contents
DTF - 26	2	Length of nonstandard label if present.
DTF - 24	1	Number of nonstandard-labeled reels if specified in the system-name.
DTF - 23	1	Counter used by the ILBDNSLO subroutine. It is initially set equal to byte DTF-24. Thereafter, it is decremented each time a nonstandard-labeled reel is processed, and so indicates the number yet to be processed.
DTF - 22	2	Maximum record length, if the file is variably blocked without an APPLY WRITE ONLY clause specified.
DTF - 20	4	Address of the PN for a USE declarative to process BOV labels. The USE statement has either the BEGINNING REEL or the BEGINNING UNIT option.
DTF - 16	4	Address of the PN for a USE declarative to process EOY labels. The USE statement has either the ENDING REEL or the ENDING UNIT option.
DTF - 12	4	Address of the PN for a USE declarative to process trailer labels. The USE statement has the ENDING FILE option.
DTF - 8	4	Address of the PN for a USE declarative to process header labels. The USE statement has the BEGINNING FILE option.
DTF - 4	1	Pre-DTF Switch. For the format of this switch, refer to "Pre-DTF Switch" in this chapter.
DTF - 3	3	Address of the PN for a USE declarative to process standard errors. Format 2 of the USE declarative is used.

Notes:

1. If any of the options concerned are not specified, the field contains 0s.
2. Phase 21 fills in only the four bytes beginning at DTF - 24.

Figure 17. Pre-DTF for DTFMT and DTFSD

Location	Length (Bytes)	Contents
DTF - n	n - 26 (9-263 bytes)	ACTUAL KEY. This field is present only for COBOL WRITE statements.
DTF - 26	8	SEEK address, in the form MEBCCCHR.
DTF - 18	2	Error bytes. This area is reserved for the DOS/VS Supervisor and assigned the name ERRBYTE. For a complete discussion, refer to <u>IBM DOS/VS Supervisor and I/O Macros</u> , Order No. GC33-5373.
DTF - 16	4	Pointer to area where extent information is saved.
DTF - 12	4	Address of the PN for a USE declarative to process trailer labels. The USE statement has the ENDING FILE option.
DTF - 8	4	Address of the PN for a USE declarative to process header labels. The USE statement has the BEGINNING FILE option.
DTF - 4	1	Pre-DTF Switch. For the format of this switch, refer to "Pre-DTF Switch" in this chapter.
DTF - 3	3	Address of the PN for a USE declarative to process standard errors. Format 2 of the USE declarative is used.

Notes:

1. If any of the options concerned are not specified, the field contains 0s.
2. Phase 21 fills in only the four bytes beginning at DTF - 16.

Figure 18. Pre-DTF for DTFDA, Random Access, Absolute Addressing

Location	Length (Bytes)	Contents
DTF - n	n - 26 (5-258 bytes)	ACTUAL KEY. This field is present only for COBOL WRITE statements. Standard C
DTF - 26	4	SEEK address, in the form TTR.
DTF - 22	3	Last extent used, in the form TTT.
DTF - 19	1	Not used.
DTF - 18	2	Error bytes. This area is reserved for the DOS/VS Supervisor and assigned the name ERRBYTE. For a complete discussion refer to <u>IBM DOS/VS Supervisor and I/O Macros</u> , Order No. GC33-5373.
DTF - 16	1	Index to the last extent used in the disk extent table in the DTF.
DTF - 15	3	Pointer to the disk extent table in the DTF.
DTF - 12	4	Address of the PN for a USE declarative to process trailer labels. The USE statement has the ENDING FILE option.
DTF - 8	4	Address of the PN for a USE declarative to process header labels. The USE statement has the BEGINNING FILE option.
DTF - 4	1	Pre-DTF switch. For the format of this switch, refer to "Pre-DTF Switch" in this chapter.
DTF - 3	3	Address of the PN for a USE declarative to process standard errors. Format 2 of the USE declarative is used.

**Notes:**  
 1. If any of the options concerned are not specified, the field contains 0s.  
 2. Phase 21 fills in only the four bytes beginning at DTF - 16.

Figure 19. Pre-DTF for DTFDA, Random Access, Relative Addressing

Location	Length (Bytes)	Contents
DTF - 31	8	SEEK address, in the form MBBCCHHR.
DTF - 23	5	Address of the next record to be read, in the form CCHHR. This area is named IDLOC.
DTF - 18	2	Error bytes. This area is reserved for the DOS/VS Supervisor and assigned the name ERRBYTE. For a complete discussion, refer to <u>IBM DOS/VS Supervisor and I/O Macros</u> , Order No. GC33-5373.
DTF - 16	4	Pointer to area where extent information is saved.
DTF - 12	4	Address of the PN for a USE declarative to process trailer labels. The USE statement has the ENDING FILE option.
DTF - 8	4	Address of the PN for a USE declarative to process header labels. The USE statement has the BEGINNING FILE option.
DTF - 4	1	Pre-DTF Switch. For the format of this switch, refer to "Pre-DTF Switch" in this chapter.
DTF - 3	3	Address of the PN for a USE declarative to process standard errors. Format 2 of the USE declarative is used.

**Notes:**  
 1. If any of the options concerned are not specified, the field contains 0s.  
 2. Phase 21 fills in only the four bytes beginning at DTF - 16.

Figure 20. Pre-DTF for DTFDA, Sequential Access, Absolute Addressing



Location	Length (Bytes)	Contents
DTF - 31	4	SEEK address, in the form TTTR.
DTF - 27	3	Last extent used, in the form TTT.
DTF - 24	1	Not used.
DTF - 23	4	Address of the next record to be read, in the form TTTR. This area is named IDLOC.
DTF - 19	1	Not used.
DTF - 18	2	Error bytes. This area is reserved for the DOS/VS Supervisor and assigned the name ERRBYTE. For a complete discussion, refer to <u>IBM DOS/VS Supervisor and I/O Macros</u> , Order No. GC33-5373.
DTF - 16	1	Index to the location in the disk extent table in the DTF where the last extent is stored.
DTF - 15	3	Pointer to the disk extent table in the DTF.
DTF - 12	4	Address of the PN for a USE declarative to process trailer labels. The USE statement has the ENDING FILE option.
DTF - 8	4	Address of the PN for a USE declarative to process header labels. The USE statement has the BEGINNING FILE option.
DTF - 4	1	Pre-DTF Switch. For the format of this switch, refer to "Pre-DTF Switch" in this chapter.
DTF - 3	3	Address of the PN for a USE declarative to process standard errors. Format 2 of the USE declarative is used.

**Notes:**

1. If any of the options concerned are not specified, the field contains 0s.
2. Phase 21 fills in only the four bytes beginning at DTF - 16.

Figure 21. Pre-DTF for DTFDA, Sequential Access Relative Addressing

Location	Length (Bytes)	Contents
DTF - 8	2	Unused.
DTF - 6	2	Displacement of record key within record.
DTF - 4	1	Pre-DTF Switch. For the format of this switch, refer to "Pre-DTF Switch" in this chapter.
DTF - 3	3	Address of the PN for a USE declarative to process standard errors. Format 2 of the USE declarative is used.

Figure 22. Pre-DTF for DTFIS

DTFIS Pre-DTF

For a file whose organization is INDEXED, eight bytes will be reserved in front of the DTF. The contents of the Pre-DTF are shown in Figure 22. Only the displacement of the record key within the record is filled in by phase 21.

DTFDU Pre-DTF

For a file that is assigned to a 3540 Diskette unit device, eight bytes will be reserved in front of the DTF. The contents of the pre-DTF are shown in Figure 22A. None of the bytes of the pre-DTF are filled in by phase 21.

Location	Length (Bytes)	Contents
DTF - 8	4	Unused
DTF - 4	1	Pre-DTF switch.
DTF - 3	3	Address of ERROR declarative PN.

Figure 22A. Pre-DTF for DTFDU

Pre-DTF Switch

This switch provides communication between the executing program and its input/output subroutines at execution time. The entire byte may be set to X'FF' to indicate that the file was closed with lock and cannot be reopened. Otherwise, the switch is used as follows:

Bit	Meaning, if ON
0	DTFSD output file. The entire DTF was saved for subsequent OPEN OUTPUT statements.
1	DTFDA or DTFSD, and OPEN I-0.
2	This call is for BOV, rather than BOP, processing. The bit is set OFF when a file is opened to indicate to the ILBDUSL0 library subroutine that BOP labels are to be processed. That subroutine sets the bit ON after BOP processing to indicate that all subsequent calls will be for BOV label processing.
3	A test must be made by the ILBDVBLO library subroutine to determine whether the present block is full. This bit is turned OFF when a file is opened and ON for all WRITES after the first.
4	DTFDA, spanned records.
5-7	Not used.

COBOL INDICATORS IN DTF'S

Certain bits within the DTFs are used as indicators at execution time. These indicator bits are the REWIND and the COBOLRWD bits in the DTFMT, which are set by phase 21, and the COBOL bits in the DTFMT and DTFSD, which are set as described in this chapter under "COBOL Bits."

REWIND and COBOLRWD

Phase 21 sets the COBOLRWD indicator (bit 6 of byte 31 of the DTFMT DTF) to 1 to force a rewind and unload at automatic end-of-volume. A setting of 0 indicates that LIOCS should check the REWIND indicator to find out what action is to be taken.

Phase 21 sets the REWIND indicator (bits 2 and 3 of byte 32 of the DTFMT DTF) to 00 to indicate rewind, to 01 for no rewind, and to 10 for rewind and unload. After the OPEN is executed, REWIND is set to 01 to ensure that the last reel is not rewound when the end-of-file condition is detected.

Before a CLOSE REEL (PEOV) is executed, REWIND is set to the specified option and COBOLRWD to 00. After an PEOV, REWIND is set to 1 and COBOLRWD to 01. Before a CLOSE for the file, REWIND is set to the specified option.

A summary of the REWIND and COBOLRWD settings and their meanings is shown in Figure 23.

SETTING		MEANING
COBOLRWD	REWIND	
1	00	Setting at compile time
1	01	OPEN NO REWIND
1	01	Setting after any OPEN until next verb
0	00	CLOSE REEL
0	01	CLOSE REEL NO REWIND
1	01	Setting after any CLOSE REEL
1	00	CLOSE
1	01	CLOSE NO REWIND
1	10	CLOSE LOCK
1	00	Setting after CLOSE or CLOSE NO REWIND

Figure 23. COBOLRWD and REWIND bits

COBOL Bits

There are six COBOL bits in DTFs. Their locations and meanings are shown in Figure 24 and the notes following it.

DTF	BYTE	BIT	MEANING, IF ON	NOTE
DTFMT	16	2	File is assigned IGN on the ASSIGN control card	1
DTFMT	16	3	Read EOF labels at CLOSE time	2
DTFMT	36	1	Do not write a user label	3
DTFSD	16	2	File is assigned IGN on the ASSGN control card	1
DTFSD	16	5	Read trailer labels at CLOSE time	2
DTFSD	16	7	COBOL End-of-Extent option	4

Figure 24. COBOL bit settings

WRITING DATA A-TEXT AND P1-TEXT

Phase 21 produces buffers, FIBs, and DTFs as a series of Data A-text elements (see "Section 5. Data Areas" for the format). Each element created is one of the following:

- DTF address element
- Secondary DTF address element
- Constant definition element
- Block address element
- FIB address element
- BL reference element
- BLL reference element

Notes:

1. The bit was set to 1 by the DOS system OPEN transient routine if the file was assigned IGN. In that case, a branch is taken to the AT END address instead of executing a READ. This test is generated by phase 51 before all READ statements.
2. COBOL always requires that user EOF labels be read at CLOSE time. Since the DOS system LIOCS routine normally reads these labels at EOF time, phase 21 sets this bit to 1 to indicate that the COBOL procedure must be followed if the file is labeled.
3. COBOL requires that user labels be written selectively (header labels, but no trailer labels, for example). Since the DOS/VS system LIOCS routine normally writes at least one user label when user labels are specified, this bit is used to indicate that the COBOL procedure is to be followed. The bit is set to 1 by the COBOL library subroutine, IIBDUSL0, if there is no user label to be written.
4. Phase 51 generates code to set this bit to 1 for all output files. When it is so set, the DOS system LIOCS routine transfers control to the end-of-extent address (INVALID KEY address) in the DTF when there are no more available extents.

For each DTF, the DTF generator creates an address element consisting of the location from field LOCCTR in COMMON and the DTF number. For each additional DTF for the same file, a secondary DTF address element, consisting of the location from field LOCCTR in COMMON and the secondary DTF number, is issued.

Constant definition elements record the values of constants to be filled into fields of the DTF.

As phase 21 creates buffers (see "Buffer Generation"), a block address element is also created for each FD. The element contains the location of the first byte of data information in the buffer and the size of the data element in words.

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

In addition to the Data A-text elements, phase 21 writes a virtual definition element of P0-text on SYS002 for each VCOW needed. Phase 51 later writes it on SYS003 as Optimization A-text input for phase 00.

DTFs For Associated Files

When a 5425, 3525 or 2560 unit record device with optional read and print features has been specified as an associated file, phase 21 generates a unique DTF for each function to be processed by the device. Routine GENDTFCD

and GENDTFPR read the Data IC-text elements created by phase 10 for each associated file and build the ASCTAB table. After all of Data IC-text has been read, routine GENASC uses the table to generate the Data A-text elements used to create the DTF's. Figure 25 shows the way in which these DTFs are chained together.

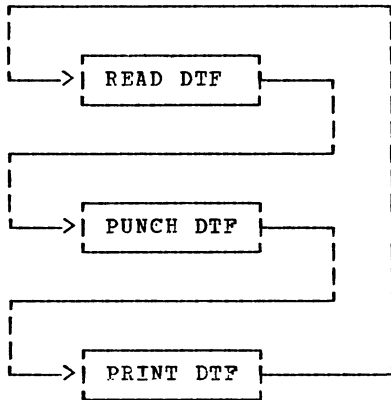


Figure 25. DTF Chaining for an Associated File with Three Functions

#### File Information Block (FIB)

Phase 21 creates the File Information Block (FIB) for VSAM files. The FIB work area is generated by means of the GENFIB macro. The AMTXT routine fills in fields of the FIB, and the BEGIN routine writes the FIB as Data A-text.

#### BUFFER GENERATION

Buffer areas are determined after all the text processing is completed. Routine

BUFGEN selects a buffer to be generated from the BUFTAB table established by the DTF generators. This routine allocates a buffer area according to the maximum size given in the table. (The size of each buffer includes any control fields required for a particular access method, for example, COUNT and KEY fields for direct access.) BUFGEN then determines the beginning of the buffer area so the data portion of each record is aligned on a doubleword boundary. After the buffer space has been allocated, the addresses of the IOAREAS are filled into the DTF, and buffer area addresses are issued as Data A-text block address elements.

Entries in the BUFTAB table indicate which files are associated by SAME AREA or SAME RECORD AREA clauses. For SAME RECORD AREA, IOAREAS are assigned in the above manner, and an additional work area is also assigned. The size of this work area is the size of the largest record area plus the size of the largest pre-record area of any of the files specified in one SAME RECORD AREA clause. The SRA generator routine builds the SRATBL, SRAMAX, and SDSRATBL tables to calculate the length and location of the SAME RECORD AREA work area in the object module. All files with a SAME AREA clause will share the same buffer area.

Using the BLTABL table formed by the DTF generators, routine PUTBL determines how to initialize the base locator or locators for the file.

Processing continues in this manner until the BUFTAB table is exhausted. After all input/output areas are generated, the tables are released, and control passes to phase 00.

Phase 25 is loaded only if the SYMDMP option has been specified on the CEL card.

The major functions of phase 25 are:

- Building the OBODOTAB table and writing it on the debug file (SYS005) if the program contains any OCCURS...DEPENDING ON clauses
- Building the DATATAB table and writing it on the debug file (SYS005).

The operations of phase 25 are described in Diagram 3.

PHASE 25 PROCESSING FOR THE DEBUG FILE

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 QITBL table which contains a COBOL dictionary pointer for every object of an OCCURS...DEPENDING ON clause
- The QRTN 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 table entry for each data item in the Data Division. There is also a DATATAB table entry for some of the compiler-generated names associated with the Report Writer feature. There is an OBODOTAB table 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 file buffer as soon as it is completed. OBODOTAB table entries are entered in the debug file on fullword boundaries. DATATAB table 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

The OBODOTAB table lists the characteristics of each unique object of an OCCURS...DEPENDING ON clause. For details on how the OBODOTAB table is built, see Diagram 4.

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 PROCRENM to process RENAMES items for data-name. PROCRENM calls ENTRDATA to move completed 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 SYS005 at end of buffer. PHASEND routine releases tables and repositions SYS004 when the dictionary processing is complete.

By the time Phase 30 (ILACBL30) is loaded into storage, almost all information on source program-names in the P0-text has been concentrated in the attributes of the dictionary entries. Supplementary information is stored in the QVAR, QFILE, INDKEY, and VALTRU tables. Phase 30 can now replace each name with its attributes, as well as add information to verb strings such as SEARCH and OPEN.

Phase 30 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 later phases.

There are four main categories of phase 30 processing, all dependent upon 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, READ and SEARCH verb strings, and verb strings with CORRESPONDING options.
- Performing any syntax analysis that requires the dictionary.

Code in the phase 30 load module is organized as follows:

- EQUATE statements for COMMON, registers, ACCESS routines, TAMER routines, and miscellaneous values.
- Phase initialization headed by the PHINIT routine and including the dictionary ACCESS routines.
- Phase control for the translation stage (PHCTRL routine).
- Main processing routines, including utility routines for acquisition and movement of data.
- Glossary-building routines, headed by the GLOSRY routine and including utility routines for placing elements in a print buffer.
- Constants.

#### PHASE 30 METHOD OF OPERATIONS

Diagram 5 shows the overall flow of phase 30 operations. Phase 30 input consists of P0-text and E-text on SYS002, the dictionary, and the QFILE, QVAR, INDKEY, and VALTRU tables in storage. Its output consists of P1-text and E-text on SYS003, DEF-text on SYS004, and the glossary on SYSLST (or SYS006 for LVL option).

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 translation, special processing is performed on READ verb strings; OPEN verb strings; ADD, SUBTRACT, and MOVE verb strings with the CORRESPONDING option; SEARCH verb strings; source program-names and special registers; and syntax errors.

#### GLOSSARY BUILDING

The PHINIT routine determines from the SYM bit of the PEZSW1 switch in COMMON whether a glossary has been requested. If it has not, the PHINIT routine branches to the TSTWRO routine. 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 WRITE-ONLY 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 a glossary has been requested, the PHINIT routine branches to the GLOSRY routine, which prints out the glossary on SYSLST (SYS006 for LVL option) and simultaneously takes the same course of action 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 placed by phase 11) and the DICND2 field (pointing to the last Data Division entry placed 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 the contents of PRLINE. When necessary, the GLOSRY routine converts numbers in the attributes from one mode to another.

TRANSLATION FROM P0-TEXT TO P1-TEXT:  
PHCTRL ROUTINE

Before the GLORET routine branches to the PHCTRL routine for the translation stage of phase 30 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 a MERGE or SORT verb, the PHCTRL routine calls the SMERGE routine to process these verbs.

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 operands. It then branches to the CORRTRN routine to break down the statement into simple statements, each using 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 CURCRD and then writes the element unchanged on SYS003. All remaining elements are also written on SYS003 unchanged.

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

If the element is a VSAM file-name, the FILENM routine initializes the KEY CLAUSE work area. This information is used by the PHCTRL routine to determine that the file specified in the KEY clause is a VSAM file and by the DATANM routine to determine that the data-name specified in the KEY clause was specified as a RECORD KEY data-name.

The READFN, 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 their verb strings are unique.

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

The processing routines perform any diagnostic analysis that requires the dictionary. When a routine detects an error that requires action parameters which 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: READFN Routine

The READFN routine checks the next P0-text element in the input buffer after a READ or RETURN verb to see if it is a file-name. If not, the READFN routine writes the verb element unchanged on SYS003 and returns to the PHCTRL routine. Phase 40 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 see if 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 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.

MERGE/SORT Verb Strings: SMERGE Routine

The SMERGE routine enters file-names specified in the USING clause into the USNGTBL table. At the end of USING clause processing, the routine produces OPEN INPUT strings for each file-name. In the case of file-names being repeated, the SMERGE routine produces an error message.

Statements with CORRESPONDING Options: CORRTN Routine

The CORRTN 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, all of which together explicitly represent the operations implied by the source statement.

Two sample CORRESPONDING P0-text statements are given in Figures 26 and 27 along with the resulting P1-text. The step numbers given on the left in these figures refer to the procedure sequence discussed below:

Step 1: If the source statement is a MOVE, the PHCTRL routine writes out only the first object hierarchy operand as P1-text. If there are others, the CORRTN routine processes them separately after the subject-object pairs. The source statements are put out so that phase 40 can perform a syntax check on them.

Step 2: Operand-1 and operand-2 are checked to make sure 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 40 finds this type of error.

If operand-1 is not an EBCDIC name but is a data operand (for example, a

literal), the CORRTN routine 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. When phase 40 finds this error, the error symbol tells it that phase 30 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. Phase 40 does not find this error.

If operand-1 is valid and 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

The word CORRESPONDING is written after the string, and then the next P0-text element is read. The string tells phase 40 that no matching pairs were put out because of an invalid operand-2.

If operand-1 is valid and 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 that 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 CORRTN routine knows which operand has the highest level dictionary pointer. Before initiating the object hierarchy search, it makes sure 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 using the HASH table (see the "Appendix A. Table and Dictionary Handling"

```

|P0-text:  ADD CORRESPONDING R TO K
|
|P1-text:
|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 26. P1-text Resulting from an ADD CORRESPONDING Option

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 clause or an OCCURS clause with a DEPENDING ON option? If it does, ignore the item.) No error symbols or messages are generated if a match is found for any of the subject hierarchy items. If no match is found, a P1-text string (verb, error symbol, preposition, error symbol) is written to tell phase 40 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, and the next P0-text element is gotten. The word CORRESPONDING tells phase 40 that the previous element was the last of a complete CORRESPONDING statement.

Step 7: If the source verb is MOVE and the next P0-text element is another operand, the procedure is started over again. For Step 1, SAME is used for operand-1, the current P0-text element is used for operand-2, and CORRESPONDING is omitted.

```

|P0-text:  MOVE CORRESPONDING A (1) TO
|          B, undefined (x), D.
|
|P1-text:
|Step 1:  MOVE CORRESPONDING A (1) TO B
|Step 5:  MOVE A (1)+ TO B+
|
|Step 5:  MOVE A (1)C TO BC
|
|Step 5:  .
|
|Step 5:  .
|
|Step 5:  .
|
|Step 5:  MOVE A (1)O TO BO
|
|Step 6:  CORRESPONDING
|
|Steps 7&1: MOVE SAME TO error
|           symbol (x)
|
|Step 2:  MOVE error symbol TO
|         error symbol
|
|Step 6:  CORRESPONDING
|
|Step 7:  MOVE SAME TO D
|
|Step 5:  MOVE A (1)+ TO D+
|
|Step 5:  MOVE A (1)C TO DC
|
|Step 5:  .
|
|Step 5:  .
|
|Step 5:  .
|
|Step 5:  MOVE A (1)O TO DO
|
|Step 6:  CORRESPONDING
|
|Note: The term "undefined" means the
|       operand is not an EBCDIC name or data
|       element; the term "error symbol" is a
|       code signaling an error.

```

Figure 27. P1-text Resulting from a MOVE CORRESPONDING Option

SEARCH Verb Strings: STSRCH Routine

For each table to be searched, there is an entry in the INDKEY table containing literals expressing 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 these literals and attributes to the SEARCH verb string.

The STSRCH routine first writes the verb element on SYS003 and then examines the element that follows. This element should

be the EBCDIC name for the table that is to be searched. 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 put out, since phase 40 detects the error later on.

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 28 shows the P1-text output for a verb string of format 1 of the SEARCH statement with the VARYING option. Figure 29 shows the output for a verb string of format 2 of the SEARCH statement.

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 control to the PHCTRL routine to handle the rest of the statement.

Determining the Uniqueness of a Name:  
SEARCH Routine

The SEARCH routine moves the source program-name it is to analyze from the input buffer to the location WKAREA. It determines if 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 looks for it in the SPCREG area, which contains the names of the 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.

Qualified 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, it is returned through the SEARCH routine to the PHCTRL routine. Its qualifiers are discarded.

<u>Phase 11 Output</u>	<u>Phase 30 Output</u>	<u>Meaning of Element</u>	<u>Processing by Phase 30</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

Figure 28. P1-Text for SEARCH Format-1 (Part 1 of 2)

Phase 11 Output	Phase 30 Output	Meaning of Element	Processing by Phase 30
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 40 convenience
	36 Attributes	Data-name for index-name-1 that does not belong to table, if specified	Name replaced by attributes found in dictionary
OR			
23 EBCDIC Name		Data-name for identifier-2 not belonging 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 40 convenience
	30 Attributes	Data-name for identifier-2 not belonging 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 28. P1-Text for SEARCH Format-1 (Part 2 of 2)

<u>Phase 11 Output</u>	<u>Phase 30 Output</u>	<u>Meaning of Element</u>	<u>Processing by Phase 30</u>
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 29. P1-Text for SEARCH Format-2

Replacing Names with Dictionary Attributes:  
GENOP Routine

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

Except for condition-names and special registers, the pointer to the dictionary entry itself is appended to these attributes. Although the dictionary does not exist after phase 30 operations, Phases 50 and 51 use the pointer as an argument in syntax analysis, and phase 60 uses it as an identification code.

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

Data-names: Data-name reference elements are generated for data-names. If the Q-routine bit in the attributes is on, the QVAR table pointer is used to find 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 verb is an OPEN verb, the GENOP routine looks 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 for the file-name.

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

Procedures-names: When the name is a procedure-name definition in a segmented program, the GENOP routine inserts a

segmentation control break after the generated P1-text element each time it encounters a section-name with a different priority.

When the name is a procedure name reference, the 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 elementary 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 elementary item, as well as to find the test-values in the VALTRU table. Figure 30 shows the P0-text input and P1-text output for a condition-string without a VALUE...THRU clause. Figure 31 shows the P0-text and P1-text for a condition-string with a VALUE...THRU clause.

Error Processing: ERROR Routine

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 on SYS003 along with P1-text, and then returns to the calling routine. The format of E-text and the manner in which diagnostic messages are later generated from it are described in the chapter on phase 70.

If CSYNTAX is specified and an error is detected, the syntax option bit in COMMON is forced on and conflicting options are forced off.

<u>Phase 11 Output</u>	<u>Phase 30 Output</u>	<u>Meaning of Element</u>	<u>Processing by Phase 30</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 ** Attri-      bytes	Conditional variable	Writes conditional variable attributes from its dictionary entry replacing its dictionary pointer with that of the condition-name
	50 06	EQUALS	Generated by GENDAT
	* ** Literal	First value to be tested for	Taken from VALTRU table entry pointed to in condition-name attributes
	54 5E	OR	Generated by GENDAT
	50 06	EQUALS	Generated by GENDAT
	* ** 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 indicating type of literal, as follows:			
	<u>Code</u>	<u>Type of Literal</u>	
	32	Numeric	
	33	Floating-point	
	34	Alphanumeric	
	39	ALL constant	
**Count indicating number of bytes in following field			

Figure 30. P1-text Written for Condition-String Without VALUE...THRU Clause

<u>Phase 11 Output</u>	<u>Phase 30 Output</u>	<u>Meaning of Element</u>	<u>Processing by Phase 30</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 ** Attri-    butes	Conditonal variable	Writes conditional variable attributes from its dictionary entry replacing its dictionary pointer with that of the condition-name
	52 00	Left parenthesis	Generated by GENDAT
	54 5C	NOT	Generated by GENDAT
	50 0A	LESS THAN	Generated by GENDAT
	* ** Literal	First value in series	Taken from VALTRU entry pointed to in condition-name attributes
	54 5D	AND	Generated by GENDAT
	54 5C	NOT	Generated by GENDAT
	50 08	GREATER THAN	Generated by GENDAT
	BB ** Literal	Last value in series	Taken from VALTRU table entry pointed to in condition-name attributes
	52 01	Right parenthesis	Generated by GENDAT
Imperative statement	Imperative statement		GENOP returns to PHCTRL to handle remaining processing
*Code indicating type of literal, as follows:			
	<u>Code</u>	<u>Type of Literal</u>	
	32	Numeric	
	33	Floating-Point	
	34	Alphanumeric	
	39	ALL constant	
**Count indicating number of bytes in following field.			

Figure 31. P1-text Written for Condition-String with VALUE...THRU Clause



Phase 40 (ILACBL40) continues the transformation of a source program Procedure Division into machine-language instructions. Its main functions are:

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

During phase 40, the compiler-generated card number of the statement currently being processed is kept in a halfword labeled CARDNO.

TRANSLATION OF P1-TEXT TO P2-TEXT

PROCEDURE-NAMES

P1-text is read by routine IDENT. If IDENT determines that an element is a procedure-name definition, it calls routine IDLHN to process the element.

VERB STRINGS

If the element encountered is a verb, a verb analyzer is called. 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 all 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, for the following reasons:

- A string is not issued unless it is free of errors. This cannot be determined until the entire string is produced.
- Sometimes the information which 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 50 or 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 statement in the chapter on phase 51 for an example of a continuation string.

The following sections give examples of phase 40 processing for several types of verbs. These examples show the general pattern of analysis for all the verbs and use all of the phase 40 tables except the SETTBL table. The SETTBL table is used to accumulate index-names or identifiers which may precede the COBOL words TO, UP, or DOWN in the SET statement.

EXAMPLES

MOVE Statement -- Subscripting

Phase 40 processing for the MOVE statement consists simply of putting out a MOVE string which gives the number of operands and names the operands. For the input elements:

MOVE A TO B

Phase 40 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:

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

Processing for this statement is illustrated by Figure 32, which shows the contents of tables built for the statement and the P2-text strings produced.

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 32. Tables and Output for a MOVE Statement

**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 table DEFSBS. (SUBSCRIPT is a special COBOL verb used only within the compiler; table DEFSBS is 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. The first SUBSCRIPT string shown in Figure 32 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 is 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 written on SYS001.

DEBUG Card

If a procedure-name is referred to on a DEBUG card, phase 40 produces a CALL string.

The output generated for debugging procedures is illustrated in Figure 33.

When routine IDLHN analyzes a PN definition, it determines from the attributes that this PN is referred to on a DEBUG card. Routine IDLHN first 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 the GN number are saved together in table DBGTLB.

When a DEBUG card is encountered, the DBGTLB 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 itself, and the GO TO statement named in the ALTER. 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 encountered, this VNTBL entry supplies the VN number for this statement's P2-text output.

Input Statement	DBGTLB 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 33. DBGTLB Entries and P2-text for DEBUG

Input Statement	VNTBL	Output
① PN1. GOTO PN3.	④ PN1 VN1	⑥ PN1. GO VN1.
② ALTER PN4 TO PROCEED TO PN6. PN2..... GO TO PN1. PN3.....	⑤ PN4 VN2	⑦ EQUATE VN1 PN3 MOVE PN6 VN2 PN2..... GO TO PN1. PN3.....
③ ALTER PN1 TO PROCEED TO PN2. PN4. GO TO PN5.  PN5..... PN6.....		⑧ MOVE PN2 TO VN1 PN4. GO VN2. EQUATE VN2 PN5 PN5..... PN6.....

Figure 34. Table Entries and Output for ALTER Statements

At execution time, each PN is assigned a cell in the PGT (Program Global Table). 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 TGT (Task Global Table); however, the contents of these cells are not permanent. When a GO TO 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 34 gives an example of phase 40 processing for the ALTER statement.

- ① In this example, the first statement read is PN1. When routine IDLHN examines the definition of PN1, it determines from the attributes that this statement is a GO TO statement referred to by an ALTER statement
- ③ (The ALTER statement follows PN3)
- ④ Then the GO verb analyzer processes the statement. It obtains a VN number, which it stores with PN1 in the VNTBL table, and it puts out two strings.
- ⑥ The GO VN1 string means that, when this branch is executed, the address to be branched to is obtained from a uniquely identified VN cell in the TGT.
- ⑦ EQUATE VN1 PN3 means that, at execution time, the initial content

of this VN-I cell is the address of PN3. Until the value is changed by an ALTER statement, the cell is unchanged, and any execution of PN1 branches 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.)

- ③ 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.
- ⑧ At execution time, this MOVE takes the address of PN2, 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 is illustrated in Figure 35. The flow of control resulting from this ALTER is shown in Figure 36.

Figure 34 illustrates a second ALTER/GO TO pair.

- ② In this case, the ALTER statement (ALTER PN4) is read first.
- ⑤ 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.

COBOL Source	Simplified Assembler code	
PN1. GO TO PN3.	L REG,VN	PN (cells of PGT)
.	BR REG	a (PN1)
.	.	a (PN2)
.	.	a (PN3)
ALTER PN1 TO PRO-	L 0,PN2	
CEED TO PN2.	ST 0,VN	
.	.	
.	.	
.	.	
		VN (cells of TGT)
		a (PN3)
		^

Figure 35. Execution of an ALTER Statement

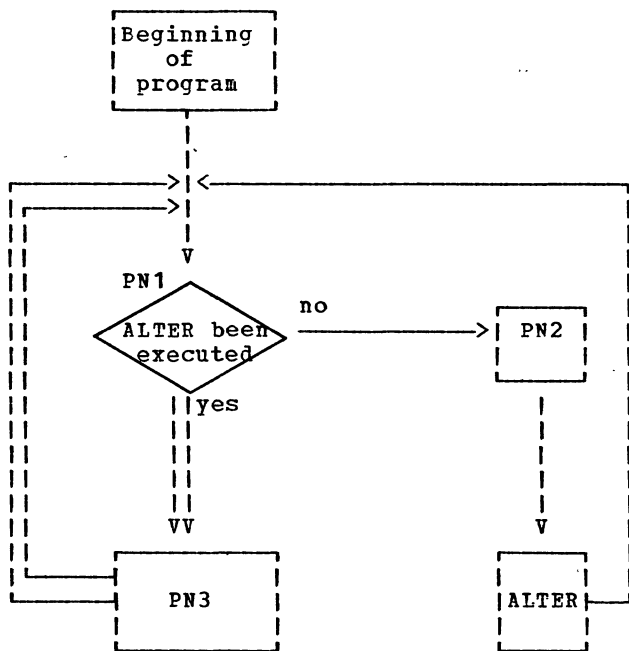


Figure 36. Flow of Control for Statements in Figure 35

**Note:** Assuming that no other ALTER or GO TO statements occur in this program, the flow of control follows the single-line path the first time through and the double-line path every time after.

Special Processing for Optimization:  
 When OPT has been specified, most PN cells are eliminated from the Program Global Table (PGT). Phases 62, 63, and 64 develop a different method for addressing these PNs, using Procedure Block base locators for this purpose. Some GN and PN cells remain unchanged by the optimizer phases. Phase 40 generates P2-text Optimization Information elements (changed to Optimization A-text elements by phase 50) to identify what type of element follows.

PERFORM Statement

The 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; that is, just before its delimiter.

Phase 40 uses the VNTBL and PFMTBL tables to keep track of the VNs. Figure 37 gives an example of the use of these tables.

Procedure Statement	VNTBL Table	PFMTBL Table	Output
SN1 SECTION. ...			SN1. ...
① SN2 SECTION. ...	② SN2 VN1	③ SN4 VN1	SN2. ...
PN3. ...			PN3. ... ⑤ GO VN1 ⑥ EQUATE VN1 SN4
④ SN4 SECTION. ...			⑦ SN4. ....
⑧ PN5. PERFORM SN1 THRU SN2.			⑨ PN5. MOVE VN1 PFMSAV1  MOVE GN1 VN1 GO SN1  GN1. MOVE PFMSAV1 VN1
PN6. ADD...			PN6. ADD...

Figure 37. Effect of a PERFORM Statement

- 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 cell VNCTR in COMMON and enters VN1 and SN2 in table VNTBL.
- ③ In table PFMTBL, 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 table PFMTBL. 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 and issues a GO string to go to VN1.
- ⑥ It issues an EQUATE string to equate VN1 to SN4.
- ⑦ It then issues the procedure-name definition for SN4.
- ⑧ When the PERFORM statement identified by PN5 is encountered, verb analyzer routine PERFORM sets up the return from the performed procedure before it issues a GO string to go to it.
- ⑨ It obtains a PFMSAV number from cell PSVCTR in COMMON and issues a MOVE string to save VN1 by moving it to PFMSAV1. (There are PFMSAV cells in the TGT to hold these values at execution time.) Then it obtains a generated procedure-name (GN) number from cell GNCTR in COMMON and issues a MOVE string to alter VN1 with GN1. This GN is issued as a GN definition for a MOVE statement to restore VN1 after the performed procedure is executed.

At execution time, the performed procedure is executed first in-line; that is, the return from it is the next sequential statement SN4. Then, when PN5 is encountered, the performed procedure is executed again. This time the return from it is GN1 (where the normal return of SN4 is restored). Then the next statement (that is, PN6) is executed.

Figure 38 gives an example of how a PERFORM statement operates at execution time. Figure 39 illustrates the flow of control for the program shown in Figure 36.

```

Source statement (COBOL):
  PERFORM SEC2 THRU SEC3.
  .
SEC1 SECTION.
  .
SEC2 SECTION.
  .
SEC3 SECTION.
  .
SEC4 SECTION.
  GO TO SEC1.

```

---

```

Resulting code (Simplified Assembler Language):
      L      0,VN          Save initial value of VN=A (SEC4)
      ST     0,PFMSAV
      L      0,GN          Set up value of VN=A (GN1)
      ST     0,VN
      L      REG,PN+4      Branch to SEC2
      B      REG
GN    L      0,PFMSAV      Re-initialize cells in TGT
      ST     0,VN
      .
SEC1  .
      .
SEC2  .
      .
SEC3  .
      .
      L      REG,VN        LOAD A (SEC4)
      B      REG          Branch
SEC4  L      REG,PN        LOAD A (SEC1)
      B      REG          Branch

```

---

```

Resulting code if OPT is specified (Simplified Assembler Language):
      L      0,VN          Save initial value of VN=A (SEC4)
      ST     0,PFMSAV
      LA     0,DISP(11)    Set up value of VN=A (GN1)
      ST     0,VN
      B      DISP(11)      Branch to SEC2
GN1   L      0,PFMSAV      Re-initialize cells in TGT
      ST     0,VN
      .
SEC1  .
      .
SEC2  .
      .
SEC3  .
      L      Reg,VN        Load A (SECA)
      B      Reg          Branch
SEC4  B      DISP(11)      SEC1

```

---

**Note:** PN and GN are cells in the Program Global Table (PGT); VN and PFMSAV are cells in the Task Global Table (TGT). The TGT and PGT are discussed in "Appendix B. Object Module."

Figure 38. Execution of a PERFORM Statement

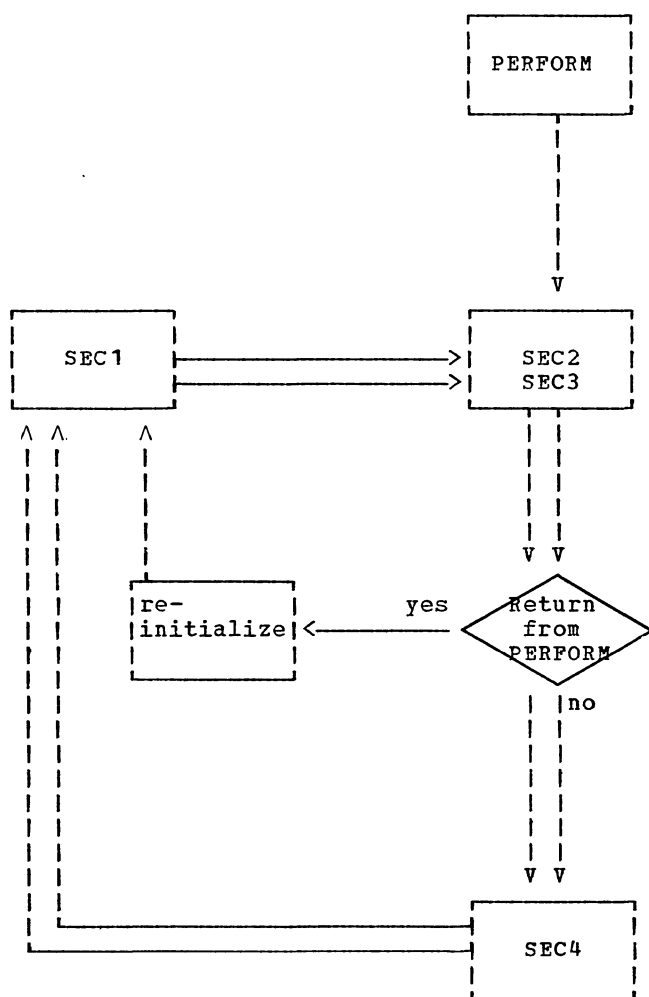


Figure 39. Flow of Control for Statements in Figure 38

**Note:** Assuming that no other procedure branching statements occur in this program, the flow of control follows the single-line path the first time through and the double-line path every time after the first.

COMPUTE Statement

Verb analyzer routine COMPUT, in conjunction 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. PNOUNT contains the nouns (that is, operands) of the arithmetic expression. PSIGNT contains the signs

(that is, 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 operators, in order of first-performed to last-performed operations, are:

- unary + and -
- \*\*
- \* and /
- + and -

For example, the statement COMPUTE X=A-B\*C requires arithmetic strings in the order:

```

MULT C B IR1
SUB IR1 A IR2
STORE IR2 X
    
```

where IR1 and IR2 are intermediate results.

Figure 40 gives an example of the use of tables PNOUNT and PSIGNT in evaluating the following statement:

```

COMPUTE X=A+(C-D/E)*F-G.
    
```

It shows which elements are added to or deleted from these tables as a result of reading a new input element, and which strings are placed in table STRING as a result of encountering this element. Each row in the figure shows table contents after processing the input element to the left

Phase 40 builds a single arithmetic string when the signs in table PSIGNT indicate that the arithmetic hierarchy of operators requires a string. To build a string, the last operator in table PSIGNT is made into a verb, the last two nouns in table PNOUNT are used as operands, and an intermediate result is appended as the temporary result. (The intermediate result is placed in table PNOUNT as an operand.) The following paragraphs give the rules for building a string. The numbers of the explanatory items correspond to the circled numbers in Figure 40.

1. If a right parenthesis is encountered, all strings up to the left parenthesis are built.
2. 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.
3. If there are no more input elements, all remaining strings are built.

Input Element	Table PNOUNT Contents	Table PSIGNT Contents	Strings Stored in Table STRING
A	A		
+	A	+	
(	A	(	
C	A C	+	
-	A C	+	
		(	
D	A C D	+	
		(	
/	A C D	+	
		(	
		-	
		/	
E	A C D E	+	
		(	
		-	
		/	
) ①	A C IR1	+	DIV E D IR1
	A IR2	+	SUB IR1 C IR2
*	A IR2	+	
		*	
F	A IR2 F	+	
		*	
- ②	A IR3	+	MULT F IR2 IR3
	IR4	-	ADD IR3 A IR4
G	IR4 G	-	
.			SUB G IR4 IR5
③			STORE IR5 X

Note: The circled numbers in the figure refer to explanations in the text.

Figure 40. Evaluation of a COMPUTE Statement

Figure 41 gives the final output from the statement.

```

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 41. 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 an operand, the number of decimal places in the result, and the presence of ROUNDED and/or ON SIZEERROR 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.

IF Statement

The IF verb analyzer, assisted by its major subroutine PFINDL, evaluates IF statements. It issues strings consisting of a relational verb, two operands to be compared, and a third operand. This last operand is a generated procedure-name (GN) to which a branch is to be made if the next statement, or the next test in a compound IF statement, is to be bypassed. The following list 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 tables PNOUNT and PSIGNT are used in evaluating arithmetic expressions in IF



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 42. Evaluation of a Nested IF Statement

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.

In addition, tables PSHTBL and PTRFLS are used in evaluating IF statements. Table PSHTBL collects branches to ELSE statements in nested IF statements. Table PTRFLS collects branches within compound IF statements.

Nested IF Statement: Figure 42 shows how table PSHTBL is used in evaluating the following statements:

```
IF A=B THEN IF C=D THEN IF E=F STOP '0'
  ELSE STOP '3' ELSE STOP '2' ELSE STOP '1'.
ADD....
```

IF and ELSE statements are paired from the inside outward. Table PSHTBL saves the procedure-names generated for branching to the ELSE statements. When an ELSE is encountered, the last generated name in the table is issued as its procedure-name definition.

SEARCH ALL Statement

The SEARCH ALL statement is executed by an object-time subroutine. (For a description of the subroutine see the publication IBM DOS/VS COBOL Subroutine

Library, Program Logic, Order No. LY28-6424.) Therefore, phase 40 does not generate statements to perform the search. Instead, it produces a parameter list for the object-time 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 43 gives an example of phase 40 output for SEARCH ALL. The example shows the P-2 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 SEARCHALL verb analyzer first receives TABLE-K and the maximum number of occurrences as P1-text input and saves them.
- 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 table KEYTBL with a flag byte of 00.

If any of the keys are found to be sterling or floating-point, a C-level E-text element is generated, and the statement is processed as if it were a format 1 of the SEARCH statement. At the

same time, WHEN conditions are processed by the IF analyzer and no special WHEN statements are generated.

After the table has been built, the SEARCHALL 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 clauses, the SEARCH verb analyzer calls other verb analyzers. However, the IF analyzer is not called to generate conditional statements. Instead, SEARCH returns control to SEARCHALL to do special WHEN processing. If the AT END and WHEN imperative statements do not provide exits from the search, phase 40 generates a GO TO NEXT SENTENCE statement.

- ③ In the example, this is accomplished via the GO GN4 statement.

Figure 44 is a generalized diagram showing the flow of execution into and out of the SEARCH ALL subroutine.

Phase 40 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. The second part of the condition should name a key matching the second entry in the KEYTBL table, and its flag byte will be set.

If keys in the WHEN clause are specified in any order other than the order of KEYTBL, E-text for a conditional error message is issued, and the SEARCH routine is given control to produce a SEARCH format-1 statement instead of SEARCH ALL.

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.

```

GO GN1.
GN2. GO PN50. (PN50 is the PN number
of NOT-FOUND)
GN3. WHEN (3)*
① TABLE-K
num-occur**
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.
SUBSCRIPT (3) TABLE-K INDEX-K
SSID1.
MOVE (2) SSID1 ENTRY-FOUND.
③ GO GN4.
GN1. GO GN3.
GN4. Next sentence.

```

\*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 40 does not produce SUBSCRIPT strings or SSIDs for them.

\*\*num-occur: If TABLE-K has a fixed number of entries, num-occur is a literal specifying the number (the value following the OCCURS clause 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 the OCCURS clause with the DEPENDING ON option.

Figure 43. Output for a SEARCH ALL Statement

SYNTAX ANALYSIS AND VERB CHECKING

Phase 40, in subroutine ERROR, issues E-text when an error is detected.

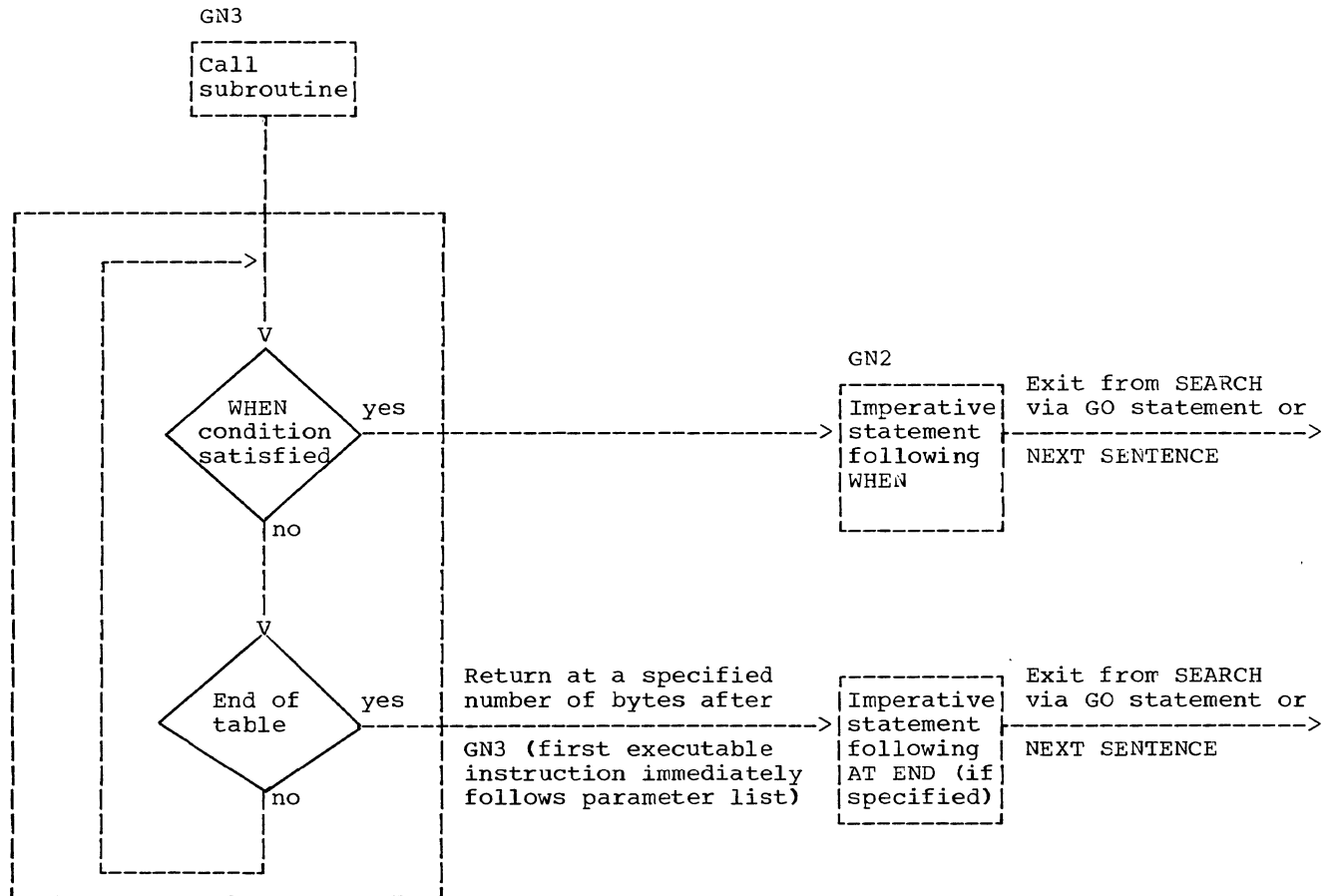
Primarily, phase 40 checks to see whether required COBOL words are present and in the correct order and whether operands are compatible with each other and in permissible format for the statement in which they are used.

In addition, it performs a limited check of the relationship between statements. For example, it detects 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 CSYNTAX is specified and an error is detected, the SYNTAX option bit in COMMON is forced on and conflicting options are forced off. If SYNTAX or CSYNTAX is specified, all error text is written to file 4.



(Solid box indicates that control flows within the SEARCH ALL object-time subroutine; blocks not within the box are executed in-line.)

Figure 44.. Flow of Execution for a SEARCH ALL Statement

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 or TIMER 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.

Branches to USE FOR DEBUGGING

After generating the return point from any USE FOR DEBUGGING procedure (other than for Procedure-name definitions), GENTIM is called.

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 ENTRIESW 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 ENTRIESW, and if TIMFLOSW is 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 routines puts the Data A-Text on SYS004.

## PHASE 50

Phase 50 (ILACBL50) reads elements of P2-text written by phase 40 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 SYS002 for phase 51, and Optimization A-text written on SYS003 for phase 60 or 62. Intermediate E-text consists of E-text passed from phase 40 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.

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 60 or 62 in eliminating duplicate references.

Input to phase 50 (a combination of P2-text and E-text) is read from file SYS001. Output is written on file SYS002, except for literal definitions written as Optimization A-text. These are written on file SYS003. (See the section "Literals and Virtuals" later in this chapter.)

Routine PH5CTL calls routine 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, Start Declaratives, and End Declaratives breaks are used for this.

If no Start Declaratives break is encountered, routine GETNXT issues the Procedure A-text for START immediately after finding the Procedure Division break. If a Start Declaratives break is encountered, the START Procedure A-text is issued after the End Declaratives break has been found.

Because 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 40; the first operand of the first string is the COBOL word FIRST, and the last element of the last string is the COBOL word LAST.

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). Routine XIS31 changes the idk field in the DOP from a subscripted or indexed reference to a data-name reference (idk represents addressing parameters; they are described fully in "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
EVAL
STORE
SUBSCRIPT
EQUATE, when in SEARCH ALL
END OF, when in SEARCH ALL
    
```

The last five are verb strings created by phase 40. 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 SYMDMP or STATE option is in effect, phase 50 generates a call to the object-time COBOL library debugging subroutine ILBDBG4 before the code to call any other object-time subroutine. For details on the object-time subroutines, see the publication, IBM DOS/VS COBOL Subroutine Library, Program Logic, Order No. LY28-6424. 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 60 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 60 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.

#### Calculating Subscripted Addresses

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:

$$\begin{aligned} &(\text{subscript1} \times \text{length1}) + (\text{subscript2} \times \text{length2}) \\ &+ (\text{subscript3} \times \text{length3}) \\ &- (\text{length1} + \text{length2} + \text{length3}) \end{aligned}$$

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:

$$(9 \times 200) + (8 \times 20) + (7 \times 2) - (200 + 20 + 2)$$

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 (how and why this table is built is described more fully later in this section). One field of the entry, called the idk field, contains a code, i, a displacement, d, and a base locator (BL) number, k. (For a description of BLs, see "Appendix B. Object Module") 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, 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.

### 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). Routine XSCOMP must first determine whether each subscript is binary. If not, code must be generated to pick 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:

1. The value of d in the XSCRPT table (the address of the subscripted field) is loaded into a register.
2. The first subscript, in binary, (X in the example) 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.
3. The result of this multiplication is added to the value of d in the first register, and this process is repeated for each subscript.
4. Phase 50 has generated a literal (referred to in this discussion as LITX) whose value is:  
$$\text{length of FIELD} + \text{length of SUBFIELD} + \text{length of ITEM}$$
5. 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.



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 execution time, the multiplications and additions will be performed for X and Z and this new value of LITX will be subtracted.

Note that, in order to arrive at a meaningful value, the entire formula must be evaluated. The intermediate results are meaningless in themselves.

Using and Optimizing Subscript References

Part of the use of the XSCRPT table is to avoid duplicate calculations for a subscripted reference. For example, consider 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. Later, when the ADD statement is executed, 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.

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 when the verb is processed, the operand is treated as though it were a data-name.

The XSSNT entry for SSID1 is now deleted, since it will never again be referenced. This is because phase 40 gave a unique number to each subscript identifier.

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. An indication of this, along with cell number, is placed in 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.

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 a 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 execution time, the index-name cells (each index-name is a one-word cell in 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.

Indexing may be direct or indirect. Direct indexing uses only index-names. Indirect indexing uses literals as increments or decrements. 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 reference, the dictionary pointers are not entered, in order 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 F.
```

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, it 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.

## 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 by phase 40:

```
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 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.

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, O

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 the processing of 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 statement example above, IR2 is found in table XINTR. 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 place) to find out 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.

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 in-line conversion or calls to COBOL library subroutines, depending on the complexity of the conversion. See the publication IBM DOS/VS COBOL Subroutine Library, Program Logic, Order No. LY28-6424.

### Register and Storage Allocation

To assign registers for an instruction, special register handling routines are used. These are:

- XRSASG 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).

If a register must be freed, routine XPREER is called. The calling routine passes the number of the register to be freed to XPREER in the XREGNO cell of the phase 50 data area. (XPREER 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 DPREER is called.

XPREER 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 TS (execution-time arithmetic temporary storage) cell where the register contents are stored, and the displacement of the cell in bytes from the

beginning of the TS area. If the value in the register was a subscript, XPREER changes the code of the idk (addressing parameter) field in the XSCRIPT 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, and the intermediate results are placed in cells of the TS area.

Space in the TS is always allotted in cells of eight bytes, 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 TS cell is made available as soon as the value in it is no longer needed.

TS 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 cell that has been assigned. When a new TS cell is assigned, this counter is incremented by one. As soon as the value in the TS cell is no longer needed, the cell number is placed in table XAVAL. The next time a TS 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 TS 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 then no longer needed. In the COMPUTE statement shown earlier, 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 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 TS, and the operation is performed from that cell.

#### Generating SRP Machine Instructions

If, at any time during processing for arithmetic verbs, scaling becomes necessary, the SRP machine instruction is generated to do left scaling or right scaling when the number of places to be shifted is even. If rounding of a number is requested, the SRP machine instruction is generated.

#### 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 data area. It is stored in the form of DCs, ready to be written out when needed. It is used for standard, frequently occurring execution-time operations, such as START.

**Direct A-text:** This is generally written as a block of instructions at a time. It is written out by routine GATXTV, which 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 (Figure 45) 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 zeros.

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.

#### 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 SYS002 and writes the literal as Optimization A-text on SYS003. At execution time, virtuals and literals are stored in the Program Global Table. By processing Optimization A-text, phase 60 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 SYS002, 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. It is incremented by 1 at phase 50 initialization because virtual 1 (MNS0) is reserved. The counter is also 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.

Virtual	Options that require the virtual
DBG0	COUNT, FLOW, STATE, SYMDMP
DBG5	SYMDMP
FLW0	FLOW
STN0	STATE
CT10	COUNT
TC00	COUNT

the only counter in COMMON handled in this manner. For all other counters, the value of the counter is incremented before being used.)

The total number of virtuals processed by phase 50 is equal to the number of virtuals required by the options in the above list plus one for MNS0. 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

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 60 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 on file SYS003; all other output from phase 50 is written on file SYS002.

Parameter Cells for the A-text Generator					
Operand-1 Parameters		Length	Element	Operand-2 Parameters	
Name	Meaning	(Bytes)		Name	Meaning
OP1	Pointer to the DOP (storage cell in the phase for a data-name operand	4	Address reference	OP2	Same for second 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.	2		XL2	Length of second operand (for SS instructions with two lengths, such as AP, SP, ZAP). The generating routine decrements value by 1.
XWC1	For arithmetic operands, specifies the operand's position in the TEMPORARY STORAGE field of the TGT. TEMPORARY STORAGE is used for arithmetic operands only.  Bytes 1 and 2: Displacement of the 8-byte slot containing this operand from the beginning of the TEMPORARY STORAGE field.  Byte 3: Number of bytes actually required by the operation. TEMPORARY STORAGE is always assigned in slots of 8 bytes, but frequently an arithmetic operand is shorter, using only a few of the low-order bytes.	3		XWC2	Same for second operand.

Figure 45. Parameter Cells for the A-Text Generator (Part 1 of 4)

Parameter Cells for the A-text Generator					
Operand-1 Parameters		Length	Element	Operand-2 Parameters	
Name	Meaning	(Bytes)		Name	Meaning
TALLY1	First operand is TALLY.	1		TALLY2	Second operand is TALLY.
BDISP1	Base (specifies a register number) and displacement of operand.	2	Base and displacement	BDISP2	Same for second operand.
RELAD1	Operand is referenced by its relative location within a field.	4	Relative address	RELAD2	Unused.
	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.				
VIRTC1	Identifying number assigned from VIRCTR when the operand is a virtual. The number begins in byte 3.	4	Virtual reference	VIRTC2	Unused.
GVIRT1	Name of virtual (used with VIRTCR).	8		GVIRT2	Unused.
XCON1	Used for two distinct types of information: A) Operand is a literal Bytes 1-16: value of literal, right adjusted Byte 17: length of literal B) Operand is a DC-type constant other than an address constant Byte 1: length of constant Bytes 2-17: value of constant, left adjusted	17	Literal reference	XCON2	Same for second operand.

Figure 45. Parameter Cells for the A-Text Generator (Part 2 of 4)

Parameter Cells for the A-text Generator					
Operand-1 Parameters		Length (Bytes)	Element	Operand-2 Parameters	
Name	Meaning			Name	Meaning
XGN1	GN number when operand is a GN reference.	2	Generated procedure-name reference	XGN2	Same for second operand.
XPN1	Operand is a PN reference Byte 1: code 00 Byte 2: priority number of PN Bytes 3 and 4: PN number	4	Procedure-name	XPN2	Same for second 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 Other Area Reference" in the Procedure A-text formats, given in "Section 5. Data Areas").	3	Global table variably located area reference	XCNTR2	Same for second operand
PLUS1	Displacement from beginning of allocated storage of a right-adjusted operand.	3		PLUS2	Same for second operand
XVN1	Operand is a VN reference Byte 1: code 00 Byte 2: priority number Bytes 3 and 4: VN number	4	Variable procedure-name reference		
BDEBG1	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").	2	Global table standard area reference	GDEBG2	Same for second operand.
BLREF1	Operand is a base locator Byte 1: type of base locator; BL, BLL, SBL ('i' field of idk) Byte 2: BL number	2	Base locator	BLREF2	Unused.

Figure 45. Parameter Cells for the A-Text Generator (Part 3 of 4)



Parameter Cells for the A-text Generator					
Operand-1 Parameters		Length	Element	Operand-2 Parameters	
Name	Meaning	(Bytes)		Name	Meaning
XREG1	Register number if first operand is a register. <b>Note:</b> When this is used, the second operand (unless it is a register also) is still considered the first nonregister operand and is placed in the operand-1 cell.	1		XREG2	Register number for second register. in an RR instruction.
IMM	Immediate field value for an SI instruction.	1			
XXREG	Register number for index register in an RX instruction.	1			

Figure 45. Parameter Cells for the A-Text Generator (Part 4 of 4)

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 in file SYS002 as P2-text. Then it determines whether the verb is MOVE, EXAMINE, or TRANSFER. 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, transferred, or read into, in which case it 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      READ
LINK     WRITE
ENTRY    REWRITE
SORT     RPTCAL
OPEN     RELEASE
CLOSE    ON
ACCEPT   STOP
RETURN
```

If the verb is one of the above, routine KILSUB is called to delete the XSCRPT table entries.

ADDITIONAL PROCESSING FOR THE OPTIMIZER OPTION (OPT)

If OPT is specified on the CBL card, phase 50 performs the following additional functions:

- Converts phase 40 Optimization Information elements (43XX) to phase 50 Optimization Information elements (COXX).
- Primes and zeros out 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 usage 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 (ILACBL51) functions in a manner similar to phase 50. Elements of text written by phase 50 are read from file SYS002. 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 file SYS001, as Optimization A-text on file SYS003, or as E-text on file SYS004. (Optimization A-text is written immediately after any Optimization A-text that was written on file SYS003 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 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 on SYS002 with an identifying prefix. This text is referred to as Intermediate E-text. Phase 51 recognizes it, discards the prefix, and writes the E-text on SYS004.

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. If any E-text was generated by Phases 10, 20, 22, or 21, ERRSEV was set by Phase 21. Otherwise, it contains a value of zero at the beginning of phase 51.

Routine ERRPRO adds one to the severity code of the current E-text element and multiplies this value by four (the code must be incremented by one because certain errors produce a severity code of 0; adding one 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 on SYS004. Note that this is the first time that E-text is separated from the other output of a phase, rather than embedded in it.

### SEGMENTATION CONTROL BREAKS

Phase 51 writes Procedure A-text on the direct access device SYS001. When phase 60 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 the source program wrote them. 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 on 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. Phase 51

stores this as an entry in SEGTL, along with the priority number of the previous segment. (For the SEGTL table format, see "Section 5. Data Areas".

The call to SEGNOTE also indicates to Phase 00 that the next time it writes on SYS001, it will be the beginning of a new segment and that 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 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 branch-in point in the object program, the rest can be eliminated. All the GNs (and the PNs, 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 number) is written. Phase 60 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 OPT has been specified on the CBL card, 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.

#### Building the PNUTBL 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 PNUTBL table for phase 60 or phase 62.

This table contains one bit for every PN definition in the program. The size of the table is determined from the cell PNCTR in COMMON, which was incremented by phase 11 every time a PN definition was created. All bits in the table are initialized to zero, and every time a PN is referenced throughout phase 51 processing, the bit in the table corresponding to the PN number is set to one. Phase 60 can thus eliminate any PN definitions whose bits are still zero.

In phase 62, the PNUTBL table is used to determine if a PN has been referenced. If it has not been referenced, no special 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 SYMDMP or the STATE option is in effect, a call to the COBOL library Save Register 14 routine (ILBDBG4), is generated for all verb analyzers (except FLOW and COUNT) which produce code branching outside the main line of the program. When routine PH5CTL encounters the READ, WRITE/REWRITE, OPEN/CLOSE, RETURN, RELEASE, and START 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 which is the operand of a CALL statement, and before any branch to a Q-routine. In addition, if the verb is a CALL, code is generated, after the BALR to the called program, to call the COBOL library debugging subroutine entry point ILBDBG3. This is done so that the subroutine can update TGT pointers. The verb analyzers for GOBACK and STOP RUN generate a call to the COBOL library

debugging subroutine entry point TC20. For a description of the COBOL library subroutines see IBM DOS/VS COBOL Subroutine Library, Program Logic, Order No. LY28-6424.

### Input/Output Verbs

Phase 51 generates the object code required for the nine 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 a LIOCS module, 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, double buffering, use of a SAME RECORD AREA clause, inclusion of a RERUN clause, use of label records and declaratives, and the type of device. For examples of the generated coding for the nine verbs discussed below, refer to Appendix D.

By convention, the following register assignments are used at execution time: R1 contains a pointer to the DTF; R2 contains IOCS's pointer to the record; R3 contains the record size; R15 contains the address of the LIOCS module or of the COBOL library subroutine called. The nine verbs and their descriptions follow.

OPEN: 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. 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.

If the user has specified the STXIT option on the CBL card and has provided an error procedure for the unit record file, phase 51 includes linkage to COBOL library subroutine ILBDABX0 in the object module. The user routine will then gain control in the event of an error on the unit-record file being opened, but only if the operator replies "cancel" to a system error message such as a data check. Phase 51 tests the PHZSW2 switch in COMMON to determine whether STXIT is in effect.

CLOSE: The code generated is the expansion of the CLOSE macro. CLOSE REEL or CLOSE UNIT, for DTFSD files, causes an expansion

of the FEOV macro instruction. For DTFDA sequential input files, subroutine ILBDCRDO is called to implement volume switching. If relative track addressing is used for DTFDA sequential input files, subroutine ILBDCR0 switches volumes. Additions may be made depending on the device and access method used. If RERUN is specified, a call to ILBDCKP0 subroutine is generated.

READ: This verb may cause either of two distinct sets of code: a GET macro expansion is generated for sequential files; a COBOL library subroutine linkage is generated for nonsequential files and in special cases.

WRITE: This verb may cause either of two distinct sets of code: a PUT macro instruction expansion is generated for sequential files; a COBOL library subroutine linkage is generated for nonsequential files and in special cases.

Note: For the READ, WRITE, OPEN and CLOSE verbs, Procedure A-text BLCHNG elements are written. This indicates to phase 60 or to phase 62, 63, and 64 that if they have permanently or temporarily loaded that BL or BLL into a register, they must reload the register or flag the register as no longer containing that BL or BLL.

SEEK: For direct-access files, the SEEK statement results in a CNTRL macro expansion.

START: The START statement for ISAM files results in the macro expansion of a SETL macro instruction with the KEY parameter or GKEY parameter.

DISPLAY: The DISPLAY statement results in the generated code for a call to the DISPLAY subroutine. The calling sequence generated is shown in IBM DOS/VS COBOL Subroutine Library, Program Logic, Order No. LY28-6424.

ACCEPT: The ACCEPT statement results in the generated code for a call to the ACCEPT subroutine. The calling sequence generated is shown in IBM DOS/VS COBOL Subroutine Library, Program Logic, Order No. LY28-6424.

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 on 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 46 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 one 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.

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 Phase 50 chapter 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 code generated for the statement.

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 phase 50 chapter.
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 which 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. It is used only if the increment is 2 or in the ON 1 case.

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 item 2, it is assigned an identifying number from ONCTR in COMMON, and it is used in a similar manner.

```

| Source Statements
|   ON 2 AND EVERY 2 UNTIL 16 MOVE
|   A TO B. ADD C TO D.
|
| P2-text Strings
|   ON 2 2 16 GN1. ①
|   MOVE (2) A B.
| GN1. ADD (2) C D.
|
| Procedure A-text
|   L 3,ONCTR1 ②
|   LA 3,1(3)
|   ST 3,ONCTR1
|   C 3,=(16) ③
|   L 2,A(GN1) ④
|   BCRNOTLO,2
|   C 3,=(2)
|   BCRLO,2 ⑤
|   X1 XSASW1,X'01' ⑥
|   CL1XSASW1,X'01'
|   BCRNOTEQ,2
|   instructions for MOVE
| GN1. instructions for ADD

```

Figure 46. Analysis of an ON Statement

### Special Considerations for Nonarithmetic Verbs

**Nonarithmetic Conversions:** In a few instances, nonarithmetic data items must be expressed in binary form during execution of the object program. 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 execution time, and places the binary value in a work area (leaving the value in the data area unchanged). The work area, rather than the data area, is then used when the value is referenced.

**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 40 passes a special verb code (see "Checking for Segmentation in Procedure Branching" in the phase 40 chapter). When phase 51 encounters this verb code, it generates a call to the COBOL library subroutine, ILBDSEM0 or ILBDSEM1 if OPT is requested. The calling sequence is given in IBM DOS/VS COBOL Subroutine Library, Program Logic, Order No. LY28-6424. This subroutine checks to see whether the necessary segment (the one containing the object of the GO TO) is in storage already, brings it into storage if it is not already there, initializes the segment, and transfers control to the named procedure.

If the operand is a PN or GN in a GO statement with the regular GO verb code, a normal branch is made whether or not the program is segmented. No test is made because phase 40 issues a regular 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 ILBDSEM0 is generated. If, at execution time, the VN is within the same segment, the subroutine will execute a normal branch.

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, ILBDSEM0, is generated. If OPT is specified, a call to the COBOL library subroutine, ILBDGDO0, is generated. This subroutine passes control to the appropriate PN. If the program is segmented, the address of entry point ILBDSEM1 is passed in register 2 to subroutine ILBDGDO0. This subroutine then determines which PN to branch to and passes control to entry point ILBDSEM1 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. The discussion is based on the DISPLAY statement shown in Figure 47. 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 the figure.

1. Phase 40 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 40 is the COBOL word FIRST, and the last operand is the COBOL word LAST. 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.)
2. ILBDDSP0 is the name of the DISPLAY COBOL library subroutine. Since it is a virtual, 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 phase

50 chapter for a discussion of how this text is produced.

3. This parameter gives the device code, which is 02 for CONSOLE. The section on ILBDDSP0 in the publication IBM DOS/VS COBOL Subroutine Library, Program Logic, Order No. LY28-6424, 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 ILBDDSP0 in the publication IBM DOS/VS COBOL Subroutine Library, Program Logic, Order No. LY28-6424, describes 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.

```

Source Statement
  DISPLAY A B C D F UPON CONSOLE.

P2-text
  DISPLAY (5) FIRST CONSOLE A B C ①
  DISPLAY (3) D E LAST

Procedure A-text
  L    15,=V(ILBDDSP0) ②
  BALR 1,15
  DC   XL2'02' ③
  DC   XL1'00' ④
  DC   XL3'000002'
  DC   AL4 (BC-DISP)
  DC   XL2'1F'
  .
  .
  DC   XL2'FFFF' ⑤
    
```

Figure 47. Analysis of a DISPLAY Verb

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 60 or phase 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 is generated. If a compare involves a field greater than 256 bytes long or if the receiving field for a move is greater than 512 bytes long, 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 long or variable in length, a call is generated to the ILBDSMV0 COBOL library subroutine.

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 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 subroutine (ILBDACLO) are

described in IBM DOS/VS COBOL Subroutine Library Program Logic, Order No. LY28-6424.

Generating Calls to the ILBDVOC0 and ILBDVIO0 COBOL Library Subroutines

The ILBDVOC0 and ILBDVIO0 COBOL object-time subroutines act as interfaces between the COBOL object program and VSAM. They use 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 60 (ILACBL60) prepares a machine language program suitable for input to the linkage editor. 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 Task Global Table (TGT) by processing counters in COMMON and calculating the displacements of items which 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 PNUTBL table. Determines storage allocation in the Program Global Table (PGT) for these items and calculates their displacements by using counters in COMMON.
- With Procedure A-text as input, generates and writes machine language instructions. If the program is segmented, groups the sections of instructions into segments and provides the appropriate linkage editor statements. If the SXREF, XREF, VERBREF, or VERBSUM option was specified, passes procedure-name and data-name definition elements, written in DEF-text, to phase 61 on SYS003 and passes procedure-name and data-name references, in REF-text, to phase 61 on SYS001.
- 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 INCLUDE statements for COBOL library subroutines to be entered at secondary entry points.
- Writes ESD, RLD, and TXT statements for all virtuals.
- Prints out compile-time statistics.

OUTPUT OF PHASE 60

The output of the phase depends on the compiler options specified by the user. The SXREF and XREF options have already been mentioned. Following are the other options which determine the output produced by phase 60:

- LISTX: Causes the TGT, Literal Pool, PGT, register assignments and a listing of the object text to be written on SYSLST (or SYS006 for LVL option).
- CLIST: Causes the TGT, Literal Pool, PGT, register assignments and a condensed object program listing to be written on SYSLST (or SYS006 for LVL option). The object program listing is limited to the card number, verb name, and address of the first instruction for each verb.
- SYM: Causes the TGT, Literal Pool, PGT, and register assignments to be written on SYSLST (or SYS006 for LVL option).
- LINK or CATAL: Causes the object program to be written on SYSLNK.
- DECK: Causes the object program to be written (punched) on SYSPCH.

The user may specify both the LINK and DECK options, in which case the object program is written on both SYSLNK and SYSPCH. He may also specify NOLINK and NODECK; in this case, he receives no executable copy of his object program.

Note that unless at least one of these eight options is in effect, phase 60 produces no output. In this event, Phase 60 returns control to phase 00 without doing any text processing unless XREF, SXREF, VERBREF, or VERBSUM has been specified. If one of these options has been specified, phase 60 processes the text to provide phase 61 with the DEF-text and REF-text elements for the cross-reference listing.

The omission of any phase 60 text processing also occurs if the SUPMAP (suppress output) option is in effect and at least one E-level or D-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 such error occurred. The ERRSEV cell is discussed further under "E-text" in the phase 51 chapter.

Phase 60 does not write object text in execution-time sequence. Rather, it instructs the linkage editor to reorder the text by assigning relative addresses in execution-time order. 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.

#### TASK GLOBAL TABLE STORAGE ALLOCATION

When phase 60 receives control, the LOCCTR contains the relative address of the Task Global Table (TGT) in the load module. It was incremented 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 -- their lengths depend 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 corresponding field in the TGT. Figure 48 lists these counters and the TGT fields to which they correspond. As the TGT is processed, a count of the displacement of the current field from the start of the TGT is kept in a register called RW1.

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 two or four 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 LISTX, CLIST, or SYM option was specified, DSPLAC calls routine MAPLOC, which prints one line at a time. If the SUPMAP condition exists, no printing is done.

If any of the debugging options 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

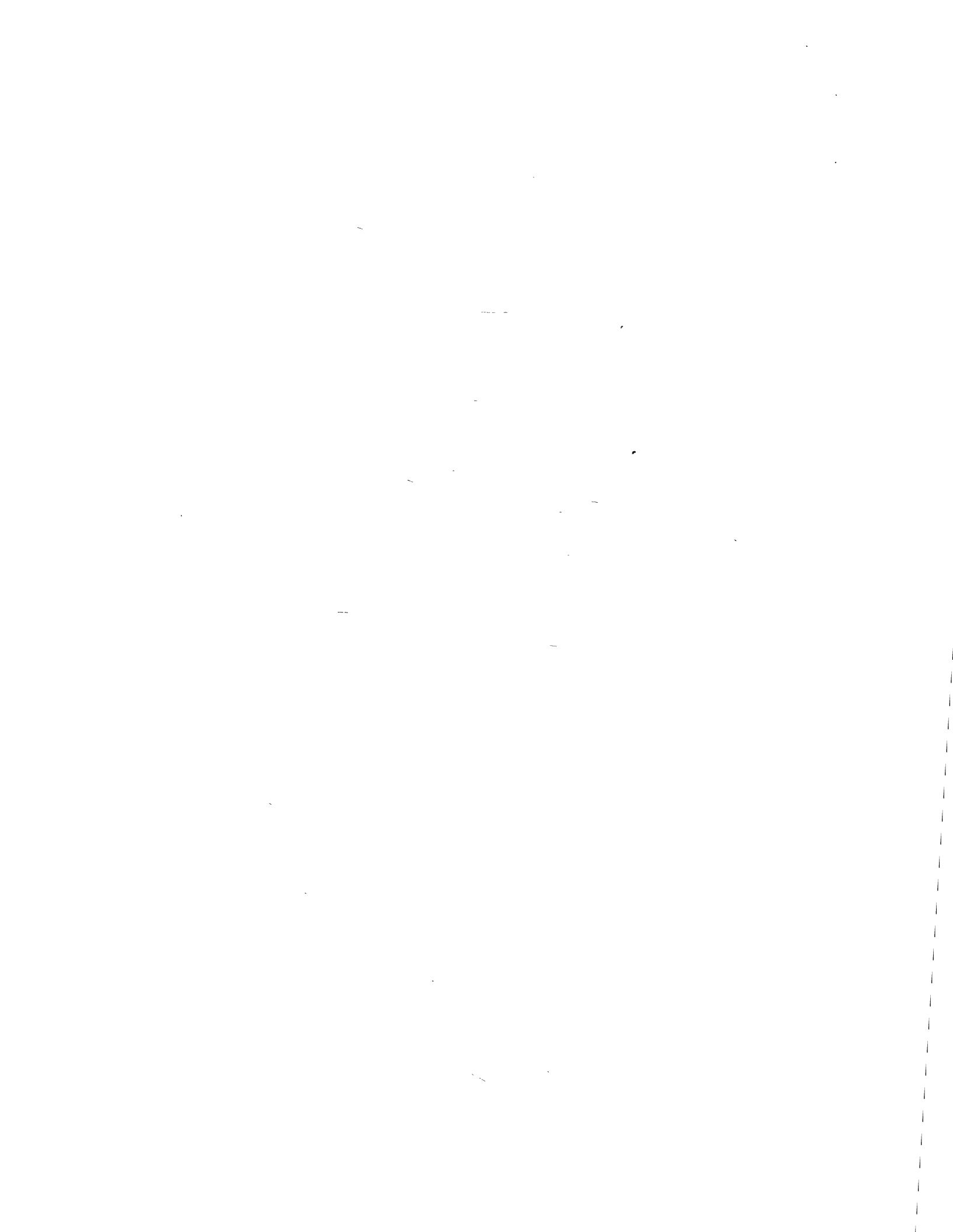
The general function of this part of phase 60 is to allocate space for the Program Global Table (PGT) in the same way that TGT storage was allocated. Before this can be done, however, the required space must be determined for the literals, virtuals, and procedure-names which reside in this table at execution time. The routines which 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. 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 generated procedure-names (GNS) to corresponding user defined procedure-names (PNS) or equate GNS to other GNS in cases where all the PNS and GNS 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 on phase 51.)
- Literal definitions, containing the actual value of the literal.
- DISPLAY literal definitions.
- Virtual definitions.

- Virtual reference definitions for error processing and input/output routines.
- Variable procedure-name (VN) definitions, if the program is segmented, for building the VNPTY table.



Counter	Contents from Earlier Phases	TGT Field	Multiplication Factor
TSMAX	Number of doublewords needed for arithmetic temporary storage	TEMPORARY STORAGE	8
TS2MAX	Number of bytes needed for nonarithmetic temporary storage	TEMPORARY STORAGE-2	1
TS3MAX	Number of bytes needed for aligning non-SYNCHRONIZED data items	TEMPORARY STORAGE-3	1
TS4MAX	Number of bytes for table-handling verbs	TEMPORARY STORAGE-4	1
BLLCTR	Number of base locators assigned to Linkage Section	BLL	4
VLCCTR	Number of variable location cell (containing current length of a variable-length field)	VLC	2
INDEX1	Number of index-names defined in files	IND	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 cells	SUBADR	4
ONCTR	Number of ON control cells	ONCTL	4
PFMCTR	Number of PERFORM control cells (for PERFORM X TIMES)	PFMCTR	4
PSVCTR	Number of PERFORM save cells	PFMSAV	4
VNLOC*	Number of variable procedure-names	VN	4
DTFNUM	Number of DTFs	DTFADR	4
XSWCTR	Number of EXHIBIT switches	XSASW	1
XSACTR	Number of bytes for EXHIBIT saved area	XSA	1
PARMAX	Number of words for parameter area	PARAM	4
RPTSAV	Number of words for report save area	RPTSAV AREA	4
CKPCTR	Number of checkpoint cells needed	CHECKPT CTR	4
IOPTRCTR	Number of pointers for SAME RECORD AREA clauses	IOPTR CELLS	4
AMICTR	Number of FIBs for VSAM files	FIB	4

\*The number of VNs in the program is passed to phase 60 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. This is because the number of VNs in the program must also be known for PGT allocation to determine the size of the VNI field in the PGT.

Figure 48. Use of Counters in COMMON To Allocate Space in the TGT for Variable-Length Fields

Virtual References Definitions: FILTBL Table

During the virtual optimization, the FILTBL table is built for the virtual names related to the input/output and error-processing routines. Virtual reference elements are not optimized and do not become part of the PGT. It is convenient to have them here, since RLDs and ESDs can be punched for them at the same time as for the other virtuals. They are placed in the DTF portion of the object module in the LIOCS module name field.

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 completely in storage for the next part of phase 60 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 VN number.

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. Then, in routine PNUPRO, the PNTBL is processed against the PNTBL. (The PNTBL table, containing one entry for each procedure-name in the program, is used only in phase 60; the PNTBL was built by phase 51. See "Building the PNTBL Table" in the chapter on phase 51 for a description of how and why this table was created.) If a PNTBL entry has a value of one, the corresponding PNTBL entry is numbered. The numbers are sequential, beginning with one. If the PNTBL entry has a value of zero, 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 zero. Once the PNTBL values have all been set, the PNTBL table is released.

Figures 49 through 51 provide an example of this processing for a program containing six PNs and six GNs. In Figure 52, the table entries are shown as they would appear after the PNTBL table 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) which process these strings.

Note: In Figures 49 through 51 the numbers to the left of the tables are A-text PN and GN numbers. They specify implicit positions in the table.

	PNTBL	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 49. PNTBL, PNTBL, and GNTBL Tables at the Beginning of Optimization Processing

Figure 50 shows the effect of a PN EQUATE string indicating that GN1 equals PN3. The referenced number of PN3 (the number found in the PNTBL entry for PN3 -- 2 in the example) is entered into the position 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 50, 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 is 5, since GN2 contains the next sequential number after PN6.

	GNTBL	
1	2	} Equates GN1 to PN3
2	0	
3	0	
4	5	} Equates GN4 and GN5 to GN2, which, it can be assumed, will be assigned a relating number of 5.
5	5	
6	0	

Figure 50. 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 51.

GNTBL	
1	2
2	5
3	6
4	5
5	5
6	7

Figure 51. GNTBL Table After the Relative Numbers Have Been Assigned

**Note:** In the object code listing, the optimized GNs are numbered sequentially, starting with 1.

Optimizing Literals and DISPLAY Literals

The literal optimizing 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 internal 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 having 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 into table LTLTBL 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 52 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

The DISPLAY literal "3" is assumed to have a length of 5, and the CONTBL literals each have a length of 1.

CONTBL	
Literals	8
	3
	9

Write text, increment RW1, and release tables

CONDIS	
DISPLAY Literals	3
	Y

LTLTBL	
0 (CONTBL Displacement of 8)	Add PGT displacements and save table
0 (CONDIS Displacement of 3)	
1 (CONTBL Displacement of 3)	
2 (CONTBL Displacement of 9)	
5 (CONDIS Displacement of Y)	
0 (CONTBL Displacement of 8)	

Figure 52. CONTBL, CONDIS, and LTLTBL Tables After Processing Literals

After the Optimization A-text file is closed, the literal pool is written on SYS1ST (SYS006 for LVL option), using the contents of the CONTBL and CONDIS tables, if the LISTX, CLIST, or SYM option was specified (no printing is done if the SUPMAP condition exists).

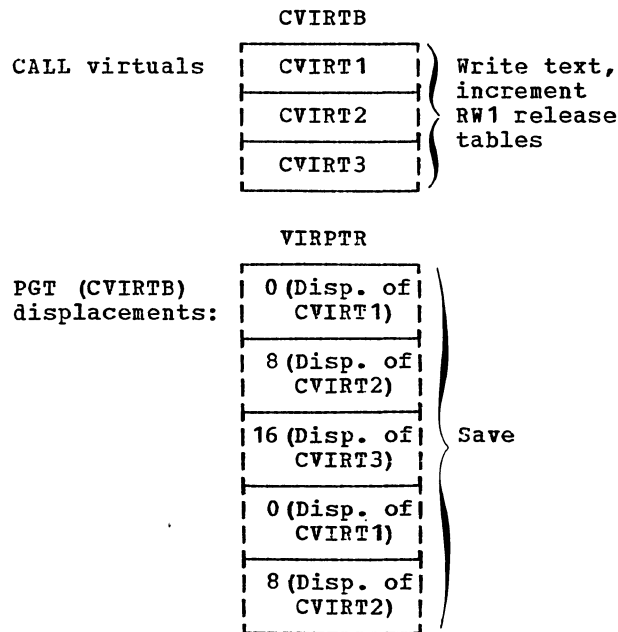
Optimizing Virtuals

The virtual optimizing 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 subroutine. The logic of this processing is similar to that of literal optimization. Two tables are built: the CVIRTB table, containing one entry for each unique virtual, and the VIRPTR table, containing one entry for each reference to a virtual.

When a virtual definition is encountered, the CVIRTB is searched for an identical entry. If none is found, the new virtual is entered in the CVIRTB table. The displacement of this virtual from the beginning of the CVIRTB is entered into the VIRPTR 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 53 shows the contents of these tables after processing Optimization A-text for a program containing the following virtuals:

CVIRT1, CVIRT2, CVIRT3, CVIRT1, CVIRT2.



Note: Each CVIRTB entry is 8 bytes long.

Figure 53. CVIRTB and VIRPTR Tables After Processing Virtuals

ALLOCATING STORAGE FOR THE PROGRAM GLOBAL TABLE

When all Optimization A-text has been read, storage is allocated for the PGT. If the program is not segmented, entries for the external symbol dictionary are created for virtuals, object text is written for virtuals and literals, and entries are made in the RLDTBL table for subsequent writing of the relocation dictionary. Counters in COMMON are set to the displacements of their corresponding PGT fields from the beginning of the PGT.

If the LISTX, CLIST, or SYM option was specified, the format of the PGT is written on SYSLST (SYS006 for LVL option) using routine MAPLOC (if the SUPMAP condition exists, this writing does not take place).

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. The number of bytes required is placed in register RW1, which is used throughout PGT allocation to hold the displacement of the field currently being processed.

VIRTUAL Allocation: After the OVERFLOW CELLS field of the PGT has been calculated, the VIRTUAL field is processed. The VIRCTR cell of COMMON is set to the displacement of the VIRTUAL field from the beginning of the PGT (this value is zero 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 RW1. The CVIRTB table is kept for use during Procedure A-text processing, when the values (EBCDIC names) in it are used to generate in-line constants for CALL statements.

The entries in the VIRPTR table are changed to contain displacements in the VIRTUAL field (see the example in Figure 41). The table is saved for subsequent use during Procedure A-text processing.

In Figure 54, the plus sign (+) means the displacement in bytes of the VIRTUAL from the beginning of the VIRTUAL field in the PGT. The values are also PGT displacements if no overflow cells are present.



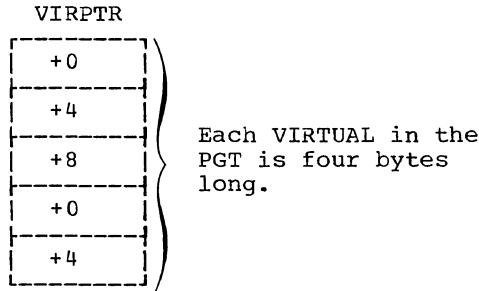


Figure 54. VIRPTR Table After VIRTUAL Allocation

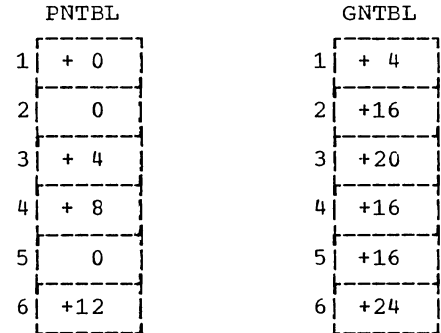


Figure 55. PNTBL Values After PGT Allocation

**PN Allocation:** After the VIRTUAL field has been processed, the value in 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 zero), no space is allocated. In the example in Figure 55, 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 zero. The total length of the PN field (16 bytes in the example) is then added to RW1.

Figure 55 shows the PNTBL table for the same program as in Figure 49 after PGT PN allocation. The table is saved for use during Procedure A-text processing, when the values it contains are used as displacements in instructions.

In Figure 55, the plus sign (+) means the displacement in bytes of the PN from the beginning of the PN field in the PGT. The values are also PGT displacements if no overflow cells are present. The numbers to the left of the tables are A-text PN and GN numbers. They specify implicit positions in the table.

**GN Allocation:** The value in 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 55, which shows the GNTBL table after this processing, for the same example as in Figures 49, 50 and 51. 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 contains 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 RW1.

The GNTBL table is saved for use during Procedure A-text processing.

**SDTFADR Allocation:** The SDTFCTR cell in COMMON is used to determine how much space is required; four bytes are reserved for each SDTF in the program. The cell is set to the value of 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. The value of VNCTR is moved to the VNILOC cell of COMMON. From VNILOC, four 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 RW1.

**LITERAL Allocation:** The displacement of the LITERAL field in the PGT (the value of 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 CONTEBL, 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 CONTEBL displacement of the literal) to the value of RW1.

The same processing occurs for LTLTBL entries that refer to CONDIS, except that the increment includes the length of the CONTEBL table. This is because DISPLAY literals are placed after internal literals in the PGT. The LTLTBL table is saved for use during Procedure A-text processing. The lengths of the CONTEBL 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, writing of object text is delayed, and therefore the tables are kept.

#### PROCEDURE A-TEXT PROCESSING

Phase 60 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 the REGMTX table (see "Execution Time Base Register Assignment" in this chapter), which is a table internal to phase 60. Base locators were assigned by phase 22.

If the LISTX compiler option is in effect, routine PUT is called to write a line of text on SYSLST (SYS006 for LVL

option) every time a complete instruction has been written. If the CLIST option is in effect, this routine is called only for each verb (if the SUPMAP condition exists, this writing does not take place).

If the SYMDMP or STATE option is in effect, Procedure A-text is used to create Debug-text which is written on file SYS002. Debug-text elements are written by the WRITE10A, WRITE20A, and WRITE30A routines. WRITE10A is called for all card numbers encountered and produces Debug-text elements which contain the card number and the contents of the LOCCTR cell when the card number was encountered. Debug-text also contains:

- Priority elements, which give the priority number of each segment in a segmented program or zero priority for a program which is not segmented,
- Discontinuity elements, which are created when phase 60 combines two sections of equal priority that have discontinuous card numbers because of an intervening section or sections of different priority,
- Segment elements which identify the last byte of each segment.

Debug text is used by phase 65 to produce debugging information for the object-time COBOL library debugging subroutines. If any of the debugging options is in effect, phase 60 builds the TGTADTBL table, which is used to pass debugging information to phase 65.

If the SXREF, XREF, VERBREF, or VERBSUM option is in effect, Procedure A-text is used to create REF-text and to write it on file SYS003. This text, containing an element for every data-name, file-name, procedure-name, and verb in the program, is used by phase 61 to produce a cross-reference listing.

Figure 56 describes the processing for each type of Procedure A-text element.

Code and Type	Action Taken
2C* card number	Store in OU6CDN, XPCDNO, and OUCCDN. If LISTX or CLIST is requested, read LISTING A-text. Used to generate an in-line constant for TRACE, which calls the DISPLAY object-time subroutine. If SYMDMP or STATE is requested, write Debug-text.
30* PN definiton	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 procedure-name	Convert the current card number into an EBCDIC constant of the form:  DC X'4' DC CL5'generated card number'
44 macro-type instruction	Use byte 2 of the element as index to a branch table. phase 60 produces the required coding or takes the required action. The contents of these elements are listed in the Procedure A-text formats of "Section 5. Data Areas."
48 op code	This element contains in machine language the first two bytes of an instruction. The first byte is the op code; the second may give condition codes, registers, or other operands. For an RR type, this element contains the complete instruction. The 2 bytes following 48 are 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 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 61 if XREF was specified.
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-I field of TGT from VNLOC cell in COMMON (see "Task Global Table Storage Allocation" in this chapter). If program is unsegmented, use VN number to compute displacement of this VN cell. If program is segmented, VNs are stored in the TGT in order of priority number, not VN number; search the VNPTY table (see "Building the VN Priority Table" in this chapter) for this element. Displacement in the VN-I field will be the same as displacement in the table.
*Indicates no object text written for this element.	
**At execution time, register 12 always points to the beginning of the PGT, 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 register 12 and register 13, respectively. The OVERFLOW cell to be used is determined from the value of the displacement; that is, a value from 4096 to 8191 uses cell 1, from 8192 to 12,227 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.	

Figure 56. Processing Procedure A-text Elements (Part 1 of 3)

Code and Type	Action Taken
58 virtual reference	Use virtual number as an index into the VIRPTR table (see "VIRTUAL Allocation" in this chapter). Table entry contains displacement of this virtual in the PGT. Use register 12** as base.
5C BL reference	This element is operand of an instruction which loads a base register. Use register 13** as base. Get the displacement of BLL or BL field in the TGT from BLCTR or BLCTR in COMMON. Use BL number to compute displacement of this cell. Update table REGMTX.
60 TGT standard area reference	Use register 13** as base. Displacement is picked up from a list of constants.
64 Global Table variable located area reference	Use register 13** as base (unless the element specifies the SDTPADR 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 48 in this chapter.
68 literal reference	The number in the text element refers to the sequential number assigned to the literal from the LTLCTR cell in COMMON. The element is used to calculate the displacement of the literal in the LITERALS field of the PGT.
6C DC definition	This element is used to create an in-line constant for a calling sequence. Bytes following the code are used to write text.
70 base and displacement	Specifies the actual register number and displacement for the instruction. Bytes following the code are used to write text.
78 address reference	Search table REGMTX on i and k (BL type and BL number). If match is found, the required BL is already in a register. Use that register as base. If no match is 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, "Global Table variably located area reference," in this table for a description of how the LOAD is generated). Displacement is d field of the element. Get card number from XPCDNO to write an element of REF-text.
7C EBCDIC data-name reference	This element always follows a 2C element. It is used for listing VERBS and paragraph-names when CLIST or LISTX is in effect. It is also used to generate an in-line constant for TRACE, which calls the DISPLAY object-time subroutine.
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 (the value of the increment). Add it to the d (displacement) field of whatever reference preceded it.
*Indicates no object text written for this element.	
**At execution time, register 12 always points to the beginning of the PGT, 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 register 12 and register 13, respectively. The OVERFLOW cell to be used is determined from the value of the displacement; that is, a value from 4096 to 8191 uses cell 1, from 8192 to 12,227 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.	

Figure 56. Processing Procedure A-text Elements (Part 2 of 3)

Code and Type	Action Taken
84 relative address	This element is used to create an in-line 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 instruction element, and must follow certain of these elements. See the list of macro-type instructions in "Section 5. Data Areas" under "Procedure A-text."
A4 incremented address	This element combines the address reference (code 78) and increment (code 80) elements into one. See those elements in this table.
B0 calling sequence displacement	Used to create an in-line TGT or PGT pointer for a call to an object-time subroutine which expects 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 61. Pick up card number from XFCDNO.
B8* file reference	Used to write an element of REF-text.
<p>*Indicates no object text written for this element.</p> <p>**At execution time, register 12 always points to the beginning of the PGT, 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 register 12 and register 13, respectively. The OVERFLOW cell to be used is determined from the value of the displacement; that is, a value from 4096 to 8191 uses cell 1, from 8192 to 12,227 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.</p>	

Figure 56. Processing Procedure A-text Elements (Part 3 of 3)

Processing in a Segmented Program

When a program is not segmented, phase 60 reads Procedure A-text in the order in which it was written. When a program is segmented, Procedure A-text is read in such an order that the procedure instructions for the root segment are processed last. The compilation of the root segment is deferred because information contained in Data A-text must be included in the root segment and Data A-text is only processed after Procedure A-text processing is completed. Although the root segment is processed last by phase 60, all object text for the root segment before the text for the nonroot (independent and overlayable) segments is written. Nonroot segments contain only procedure instructions. The TGT, PGT, Report Writer routines, Q-routines, and data area for the entire program are included in the root segment, which is resident in storage throughout program execution.

If the program is segmented, the value of the LOCCTR cell in COMMON is saved, and LOCCTR itself is set to zero. LOCCTR is again set to zero every time the processing of a new segment begins. This is because LOCCTR points to a location relative to the beginning of the object module; since each segment is a separate object module, each begins at a relative address of zero.

Procedure A-text is read from the direct-access file SYS001 using the segment priority table (SEGTL). For a description of how this table is built, see "Segmentation Control Breaks" in the chapter on phase 51. The table format is given in "Section 5. Data Areas."

In phase 11, the priority numbers of all sections in the root segment were set to zero. If the SEGMENT-LIMIT clause was specified, the root segment consists of all sections whose priority number is less than the value of SEGMENT-LIMIT. If SEGMENT-LIMIT was not specified, the root

segment consists of all segments whose priority is less than 50.

Routine SEGPROC searches the SEGTLB table for the first entry whose priority is not 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 (more information on this routine is in the phase 00 chapter 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 SEGTLB 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. A segment may consist of one or more sections. There is a SEGTLB entry for every section whose priority differs from that of the section preceding it.

When all sections of one priority have been processed, the SEGTLB table is searched for another nonzero priority, and the process is repeated (LOCCTR is set to zero each time). 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 file SYS002 for phase 65. As machine instructions are generated, object text for the nonroot segments is written temporarily on work file SYS004. This is done so that the root segment can be written first on the output file.

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 file SYS001 by finding all entries of zero priority in the SEGTLB table.

Object text for the root segment is written directly into the output file. The object text for the nonroot segments is then copied from SYS004 on the output file.

No procedure or verb names are printed on the listing. Instead, the card number is printed, and if a card contains more than one verb, the object code for each verb is accompanied by a number beginning with 1 for each card.

#### 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, one is permanently assigned to the beginning of the Working-Storage Section, and the rest to files, in the order in which PDs occurred. If any registers are still left, they are assigned to the rest of Working Storage (if there is any). If the LISTX, SYM, or CLIST option was specified, a list of permanently assigned registers and the BLs (base locators) associated with them is written on SYSLST (or SYS006 for LVL option). 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 60 refers to and updates table REGMTX (internal to phase 60), 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 contains the necessary base locator, an instruction is generated to load the base locator (which is stored in the TGT) into temporary 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 macro-instruction type A-text elements produced by Phase 50 or 51. (See Figure 56 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).

File SYS004, 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, VERBREF, or VERBSUM option was specified. Figure 57 illustrates the contents of this file when it is read by phase 60. Figure 58 describes how each type of element is processed.

Beginning of File	
Text Description	Written By
DATA A-text	Phase 21
DEF-text (for data-names and file-names)	Phase 21*
E-text (generated by Phases 10, 20, 21, and 22)	Phase 21**
DEF-text (for procedure-names)	Phase 30*
E-text (generated by Phases 12, 11, 30, 40, 50, and 51)	Phase 51
End of File	
*Generated only if the SXREF, XREF, VERBREF, or VERBSUM option is in effect.	
**Phase 21 intermixed these first three texts on SYS004. There are no separations between texts generated by different phases. The texts are distinguishable solely by their code, which is the first byte of each element.	

Figure 57. Contents of SYS004 When Read by Phase 60

PROCESSING THE RLDTBL TABLE

After an end-of-file condition has been reached on file SYS004, the RLDTBL table is processed. Indirect address constants are resolved and then the table is 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. See "GETALL routine" in the "Appendix A. Table and Dictionary Handling."

INITIALIZATION ROUTINES

After the RLDTBL table has been processed, the initialization coding is generated. The three initialization routines, in the order in which phase 60 writes them, are INIT2, INIT3, and INIT1. All three are resident in the root segment if the program is segmented. INIT1 sets up address constants for the program's TGT, PGT, and first executable instruction, and for the three initialization routines. It then transfers control to INIT2 if the program is a subprogram, or to INIT3 if it is not. INIT2 is executed if the program is a subprogram or is entered at a secondary entry point. It establishes standard subroutine linkage. Control then passes to INIT3, which sets base registers, relocates addresses to absolute values, and transfers control to the proper point in the program to begin the execution. (For a fuller description of these three routines, including the generated code, see "Appendix B. Object Module.")

Code	Type	Action Taken
00	E-text	<p>All E-text is built into a table called ERRTBL, which is passed to phase 70. Phase 60 does not process the E-text. If the ERRTBL table overflows the space allocated to it, it is written on SYS003.</p> <p>There are two types of E-text elements: message definition 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 on phase 70). Phase 60 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, which is of variable length, and the length is determined from the value of the second byte. See "Section 5. Data Areas" for format of E-text and the ERRTBL format.</p>
04	SDF address	<p>Generate an RLDTBL entry which will cause the address of the SDF to be placed in the correct cell of the SDFADR field of the PGT at execution time. Get displacement of the SDFADR field from the SDF number to compute displacement of cell. Text element contains the value (relative address of the SDF) to be placed in the PGT cell.</p>
08	DTF address	<p>Generate an RDLTBL entry which will cause the address of the DTF to be placed in the correct cell of the DTFADDR field of the TGT at execution time. Get displacement of the DTFADDR field from cell DTFNO (see Figure 48 in this chapter) and use the DTF number to compute displacement of cell. Text element contains the value (relative address of the DTF) to be placed in the cell.</p>
0C	Block address	<p>Generate an RLDTBL entry which 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 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 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 RDLTBL entry is made. The second RDLTBL entry will cause the buffer address + 4096 (TGT addresses are in bytes) 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 58. Processing Data A-text, E-text, and DEF-text (Part 1 of 2)



Code	Type	Action Taken
24-	Working-Storage Section address	<p>Generate an RLDTBL entry which 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 48 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.</p> <p>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 + 4096 (TGT addresses are in bytes) to be placed in the next BL cell.</p>
28	Constant definition	Write object text which 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 SDTFs and DTFs and to initialize data items for which a VALUE clause was specified.
2C	Address constant definition	Generate an RLDTBL table entry which will permit the linkage editor to insert the desired address at the correct location.
30		(Not used)
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 56). When phase 60 generates the code for INIT3 (one of the initialization routines), it uses the QTBL table to generate a call to every Q-routine in order to initialize the data and table areas affected by OCCURS...DEPENDING ON data items.
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	These elements are present only if the SXREF, XREF, VERBREF, or VERBSUM option was specified. Each element is written out as it is encountered on file SYS001, to be read by phase 61. The phase 61 chapter describes how these elements are used.
4C	Procedure name DEF-text	These elements are present only if the SXREF, VERBREF, XREF, or VERBSUM option was specified. Each element is written out as it is encountered on file SYS003, to be read by phase 61. The chapter on phase 61 describes how these elements are to be used.

Figure 58. Processing Data A-text, E-text, and DEF-text (Part 2 of 2)

## PHASE 65

The function of phase 65 (ILACBL65) is to produce debugging information which is used by object-time COBOL library debugging subroutines. For information about the object-time COBOL library subroutines, see the publication IBM DOS/VS COBOL Subroutine Library, Program Logic, Order No. LY28-6424. The phase is given control only if the flow trace (FLOW), statement number (STATE), or symbolic debug (SYMDMP) compiler options are specified by the user on the CBL card. 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 6.

### PROCESSING THE FLOW OPTION

If FLOW is specified, phase 65 obtains the number of traces requested (n[n]) from the FLOWSZ cell in COMMON. The number is stored in the first byte of the DEBUG TABLE in the TGT. Phase 65 allocates space for the FLOW trace table in the object module following INIT3 and saves the address of the beginning of the table in the DEBUG TABLE in the TGT. If the number of traces requested is zero, no space is allocated. For the FLOW option, the PNCHSW routine writes the flow trace information in the DEBUG TABLE and the END card. Further processing for the FLOW option is discussed in "Final Processing" later in the chapter.

### COMMON PROCESSING FOR THE STATE AND THE SYMDMP OPTIONS

For the STATE or the SYMDMP option, phase 60 created Debug-text, which is used by phase 65 to produce tables that provide information needed by the STATE or SYMDMP COBOL library subroutines. Phase 65 builds the PROCTAB and SEGINDX tables for either option. For the STATE option, the PROCTAB and SEGINDX tables are written in the object module; for the SYMDMP option, they are written on the debug file. The CARDINDX, PROCINDX, and PROSUM tables are created only for the SYMDMP option and are written on the debug file. The OBODOTAB and DATATAB tables have already been created for the SYMDMP option by phase 25.

### PROCESSING DEBUG-TEXT

Routines GETF2 and RDF2 locate and read the Debug-text, which is passed to Phase 65 on file SYS002. Debug-text consists of the following elements:

- CARDLOC elements (10), which contain COBOL source card numbers, a switch to indicate the presence or absence of a verb on the card, and the contents of LOCCTR in COMMON when the card number element was read by phase 60.
- ENDSEG elements (20), which signal the end of a segment.
- SEGMENT elements (30), which signal the beginning of a segment.
- DISCONTINUITY elements (40), which signal a discontinuity in the card numbers of the source program resulting from combining two sections of equal priority with intervening section(s) of different priority.

The Debug-text elements are directly involved in the creation of the PROCTAB, SEGINDX, and CARDINDX tables.

The F2PROCS branch table is used to branch to one of four routines which control the processing for the elements. Routine TENPROC controls processing for CARDLOC elements; routine TWENPROC controls processing for ENDSEG elements; routine SEGINDX controls processing for Segment elements; and routine FRTYPROC controls processing for Discontinuity elements.

### BUILDING THE PROCTAB TABLE

Routine TENPROC builds the PROCTAB entries from the information in the CARDLOC elements. Each PROCTAB entry contains the relative address of the first instruction generated for the card and verb number in the entry. Phase 65 divides any program or segment which exceeds 64K bytes in size into program fragments less than 64K bytes in length. Each segment also begins a new program fragment.

#### BUILDING THE SEGINDX TABLE

A SEGINDX entry is created for each fragment of the program.

If routine TENPROC determines that the code generated for the last verb causes the current fragment to exceed the maximum size of a fragment (64K bytes), it calls routine GTEQ10K to handle the processing for end of the fragment. Routine GTEQ10K calls routine SNF to start the new fragment make a SEGINDX table entry for the old fragment, and begin collecting information for the next SEGINDX table entry.

In a segmented program, the end of a segment or, in a non-segmented program, the end of the Procedure Division is signalled by an ENDSEG element (20). When routine RDF2 reads an ENDSEG element, it calls routine TWENPROC to process the end of the segment.

#### 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 in writing the PROCTAB and SEGINDX tables in the object module. The TXPNCH routine writes the PROCTAB table in the object module following either INIT3 or the Flow Trace table if space has been allocated for it. At end of file on SYS002, routine EOF2 writes the SEGINDX table in ascending order of priority after the PROCTAB table in the object module. The addresses of the beginnings 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 the chapter.

#### FURTHER PROCESSING FOR THE SYMDMP OPTION

If SYMDMP is specified, phase 65 builds the CARDINDX, PROCINDX, and PROGSUM tables for the debug file. Processing for the CARDINDX and PROCINDX tables occurs in conjunction with processing for the PROCTAB and SEGINDX tables. The PROGSUM table is processed after the other tables have been written on the debug file.

#### 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 COBOL source card numbers at the end of each non-contiguous section.

When routine RDF2 reads a Discontinuity element (40), it branches to routine FRTYPROC which sets a switch indicating that special processing is to be done for the end-of-section.

CARDINDX entries are created for discontinuity within segments and for each program segment.

#### BUILDING THE PROCINDX TABLE

Before routine TXPNCH moves a PROCTAB element into the SYS005 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.

#### DEBUG FILE PROCESSING

The TXPNCH routine writes the PROCTAB table on the debug file at the beginning of a new block.

At end of file on SYS002 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 and the SEGINDX table in order of ascending priority.

Routine EOFON2 then moves the CARDINDX, SEGINDX, and PROCINDX tables (in that order) to the buffer for the debug file (SYS005). (These tables are written on the debug file, beginning at a new block.) Routine EOFON2 saves the displacement within the buffer of the start of the SEGINDX and PROCINDX tables in the SEGDISPL and PROCDISPL save areas, respectively.

This information becomes part of the PROGSUM table.

Routine EOFON2 then calls phase 00 to write the tables and note those blocks which contain the beginning of a table. Routine EOFON2 saves the note information in the CARDNOTE, SEGNOTE, and PROCNOTE save areas for the PROGSUM table.

All the information gathered by routine EOFON2 is entered in the PROGSUM table.

If the debug file is located on disk, the first 512-byte record is read back into the buffer, and the PROGSUM table is moved into the first 84-byte field. The record is then rewritten on the disk.

If the debug file is located on tape, routine ENDOFTBL writes the end of file mark, and repositions both SYS005 and SYS002 to the first record. It reads the first 512-byte record of SYS005 into the buffer and inserts the PROGSUM table in the first 84 bytes. It writes the buffer on file SYS002 and copies the remainder of file SYS005 to file SYS002. Then it recopies file SYS002 onto file SYS005.

#### FINAL PROCESSING

For any of the options, routine PNCHSW 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. 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.

For segmented programs the root segment, written on SYSLNK by phase 60, is completed by phase 65; the independent segments are copied on SYSLNK from SYS004 where they were written by phase 60. For non-segmented programs that use the SORT/MERGE verb, phase 65 writes PHASE and text cards for a dummy SORT/MERGE subroutine load point on SYSLNK. For segmented programs that use the SORT/MERGE verb, the PHASE and INCLUDE cards are copied on SYSLNK from SYS004 where they were written by phase 60.

Routine ENDCODE releases all tables and determines whether the compiler options specified by the user require phase 00 to call phase 61 or phase 70 or whether the compilation is complete. Phase 61 is called if the SXREF, VERBREF, VERBSUM or XREF option was specified. Phase 70 is called if the highest severity message produced during compilation matches the user's specification (FLAGE or FLAGW).

Phase 65 returns control to phase 00 either to call the next phase or to process for the end of the job.

PHASE 62

Phase 62 is first phase of the three phases that make up the optimizer section of the compiler. It begins the work necessary to produce the machine language program optimized for procedure name addressability and for register usage and suitable for input the linkage editor.

The optimizer phases of the compiler (phases 62, 63, and 64) employ a method for addressing most procedure-names (PNs and GNs) in the completed machine language program which differs from the method employed by phase 60. Phase 60 places the address of the definition point of each referenced procedure-name (PNs and GNs) into the PROCEDURE NAME and GENERATED NAME cells of the Program Global Table (PGT). Each time that one of these procedure-names is referenced, a load instruction of the PN or GN address is generated. This instruction is followed by an RR-type branch instruction.

The optimizer phases divide the Procedure Division code into blocks of approximately 4095 bytes in length. These blocks are referred to as Procedure Blocks. Procedure-names (PNs and GNs) are addressed as displacements added to a base register (register 11) containing the address of the beginning of a Procedure Block. Register 11 must be loaded with the Procedure Block address when addressability is needed for the Procedure Block. Addressability must be established only when a reference is made to a PN or GN whose Procedure Block address is not the same as the one currently contained in register 11, or whenever the contents of register 11 cannot be known (see "Phase 5 Optimization Elements" under "Procedure A-text" in "Section 5. Data Areas").

The optimizer phases, eliminate the need to generate most of the instructions to load the addresses of the PNs and GNs from the PGT and instead of the RR-type branch instructions generate RX-type branch instructions. The optimizer phases also eliminate the need for most PROCEDURE NAME and GENERATED NAME cells from the PGT.

The optimizer phases optimize load instructions for base locators (BLs, BLLs, and SBLs) in the TGT and the PGT and OVERFLOW cells by permanently loading the OVERFLOW cells and then the most frequently used base locators into registers 6 through 10. Since phases 62, 63 and 64 eliminate most of the GN and PN cells, there are fewer OVERFLOW cells in a program where OPT

has been specified than there are in a program without the OPT option. Other base locators are loaded into register 14 or register 15 on a temporary basis. See "Optimizing Register Assignments" below.

Phase 62 is divided into several parts, each of which performs specific functions. The functions are:

- Determining object program storage allocation for the Task Global Table (TGT) by processing counters in COMMON and calculating the displacements of items which reside in the TGT at execution time.
- Optimizing literals and virtuals by processing Optimization A-text. Determining storage allocation in the Program Global Table (PGT) for these items and for the procedure name (PN) and generated procedure name (GN) cells.
- If the program is segmented, reading the segments in order of ascending priority.
- Determining approximate object program storage for the Procedure Division by reading Procedure A-text and calculating the Procedure Block number in which each procedure-name (PN or GN) is located.
- Optimizing usage for both permanent and temporary register assignments.

The operations of Phase 62 are described in Diagram 7.

OUTPUT OF PHASE 62

The output of phase 62 depends on the options specified by the user. The LISTX, CLIST, LINK (or CATAL), and DECK options are processed by phase 62 in the same way as they are processed by phase 60.

If the LISTX, CLIST, or SYM options have been specified, phase 62 causes the TGT, Literal Pool, PGT, and register assignments to be written on SYSLST (or SYS006 for LVL option). Phase 64 causes the object program listings produced for the LISTX and CLIST options to be written on SYSLST (or SYS006 for LVL option). Phase 64 also passes XREF-text to phase 61 if the user

specified the SXREF, XREF, VERBREF, or VERBSUM option.

For details see "Output of Phase 60" in the chapter "Phase 60."

#### ALLOCATING STORAGE FOR THE TASK GLOBAL TABLE (TGT)

Phase 62 calculates the length of the Task Global Table (TGT) in the same manner as phase 60 does. It receives the relative address of the TGT from phases 22 and 21 in LOCCTR; and it computes the length of the variable-length fields of the TGT from the counters in COMMON. In addition to the fields which both phase 60 and phase 62 allocate, phase 62 initializes a fullword to the address of the first Procedure Block cell in the PGT. For more details on the general functioning of this part of phase 62, see "Task Global Table Storage Allocation" in the chapter "Phase 60." See also Figure 48 for a description of the counters in COMMON used by this phase.

#### OPTIMIZING AND ALLOCATING STORAGE FOR THE PROGRAM GLOBAL TABLE (PGT)

Phase 62 optimizes the fields of the Program Global Table (PGT) by eliminating duplications which may have been generated by earlier phases. It also calculates the lengths of the optimized fields so that storage requirements for the PGT may be determined.

#### OPTIMIZING AND BUILDING TABLES

While optimizing for the PGT, Phase 62 reads Optimization A-text from file SYS003 and merges its information with information from tables and counters built by earlier phases. This information is used to:

- Process DTF virtuals.
- Build the VN priority (VNPTY) table.
- Optimize and calculate storage requirements for literals, DISPLAY literals, and virtuals.
- Build the BLVNTBL, PNATBL, and GNATBL tables.

#### DTF Virtuals and VN Priority Table

Phase 62 builds the FILTBL table to store the virtuals needed for input/output and error-processing routines as well as the VN priority (VNPTY) table in the same way that phase 60 does. For details see "Virtual References Definitions: FILTBL Table" and "Building the VN Table" in the chapter "Phase 60."

#### Optimizing Literals and DISPLAY Literals

Phase 62 builds the CONTEL, CONDIS, and LTLTBL tables to optimize for internal literals and DISPLAY literals in the same way as phase 60 does. The CONTEL table is used to eliminate duplicate internal literals; and the CONDIS table is used to eliminate duplicate DISPLAY literals. The LTLTBL table contains the displacements of individual literals within their respective tables. It is used later in the phase to calculate the relative address of each literal and DISPLAY literal in the PGT of the completed object program. For details see "Optimizing Literals and DISPLAY Literals" in the chapter "Phase 60."

#### Optimizing Virtuals

Phase 62 builds the CVIRTB table and the VIRPTR table to optimize for virtuals (that is, names in EBCDIC of called programs or of object-time COBOL library subroutines) in the same way as phase 60 does. The CVIRTB table is used to eliminate duplicate virtuals, and the VIRPTR table contains the displacements of individual virtuals in the CVIRTB table. It is used later in the phase to calculate the relative address of each virtual in the PGT of the completed object program. For details see "Optimizing Virtuals" in the chapter "Phase 60."

#### Processing for PNs and GNs

Phase 62 builds four tables for PN and GN processing for the PGT. These are the BLVNTBL, VNPNTBL, PNATBL and GNATBL tables. Routine VNPNSCRT builds the VNPNTBL table from the VN EQUATE PN or VN EQUATE GN elements of Optimization A-text.

Routine GNVNRTN builds the BLVNTBL table from GN and VN perform elements. Routine PGNARTN builds the PNATBL and GNATBL

tables. These tables list the PNs and GNs for which address cells are required in the PGT.

#### ALLOCATING STORAGE FOR THE PGT

When all Optimization A-text has been read, storage is allocated for the PGT. ESD cards and TXT cards are produced for virtuals and literals. Counters in COMMON are set to the displacements of their corresponding PGT fields from the beginning of the PGT.

If the LISTX, CLIST or SYM options are specified, the format of the PGT is written on SYSLST (SYS006 for LVL option) using routine MAPLOC (if the SUPMAP condition exists, this writing does not take place).

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, this part of phase 62 cannot 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 determined. The number of bytes required is placed in register RW1, which is used throughout PGT allocation to hold the displacement of the field currently being processed.

VIRTUAL Allocation: Phase 62 allocates storage for the VIRTUAL cells field of the PGT in the same way that Phase 60 does. The displacement of the VIRTUAL cells field from the beginning of the PGT is placed in the VIRCTR cell in COMMON; the calculated length of the field is added to the contents of register RW1; the displacements of the individual virtuals in the VIRTUAL field are placed in their respective VIRPTR table entries. For details see "VIRTUAL Allocation" in the chapter "Phase 60."

PN Allocation: After the VIRTUAL cells field has been processed, the value in register RW1 is the displacement of the PN field in the PGT. This value is saved in the RPNCNTR cell of COMMON.

Only those PNs which follow TO PROCEED TO in an ALTER statement and section-names defined in a USE statement in the Declaratives Section require PN cells in the PGT. Phase 51 sets the RPNCNTR counter

in COMMON to the number of cells required. Phase 62 uses the RPNCNTR 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 cells 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 RGNCTR cell in COMMON.

Only those GNs which 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.

SDTFADR Allocation: The SDTFADR cell in COMMON is used to determine the storage requirements for the SDTFADR ADDRESS cells of the PGT. Four bytes are reserved for each SDTF in the program. The cell is set to the value of register RW1, and register 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. The value of VNCTR is moved to the VNILOC cell of COMMON. Four VNILOC, four 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 RW1.

LITERAL Allocation: Phase 62 allocates storage for the LITERAL field in the PGT in the same way that Phase 60 does. The displacement of the LITERAL field is placed in the LTLCTR cell in COMMON; the calculated length of the field is added to the contents of RW1; and the LTLTBL table is saved for Procedure A-text processing. For details, see "LITERAL Allocation" in the chapter "Phase 60."

PROCEDURE BLOCK Allocation: Phase 62 does not allocate storage for the PROCEDURE BLOCK cells 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. For details on the phase 62 optimization of Procedure A-text, 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 cells field in the PGT. This value is saved in the PRBLDISP cell of COMMON. After Procedure A-text processing, phase 62, using the PROCBL counter, allocates one 4-byte field 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, routine REGMV1 first 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, routine BLSRCH searches the BLUSTBL table, built by phases 50 and 51, to determine the base locator most frequently referred to. That base locator is assigned to the next unassigned register. The process is repeated until each of registers 6 through 9 have been assigned. After Procedure A-text has been read, register 10 is assigned to the next most frequently referenced 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, routines ENTDRP and ENTDRPL 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 referenced 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 referenced 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 referenced PN, a GN, or the entry point of a new program segment.
- The base locator which 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 ELCHNG element.

Figure 59 exemplifies the optimizing process for base locator assignments to registers 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.



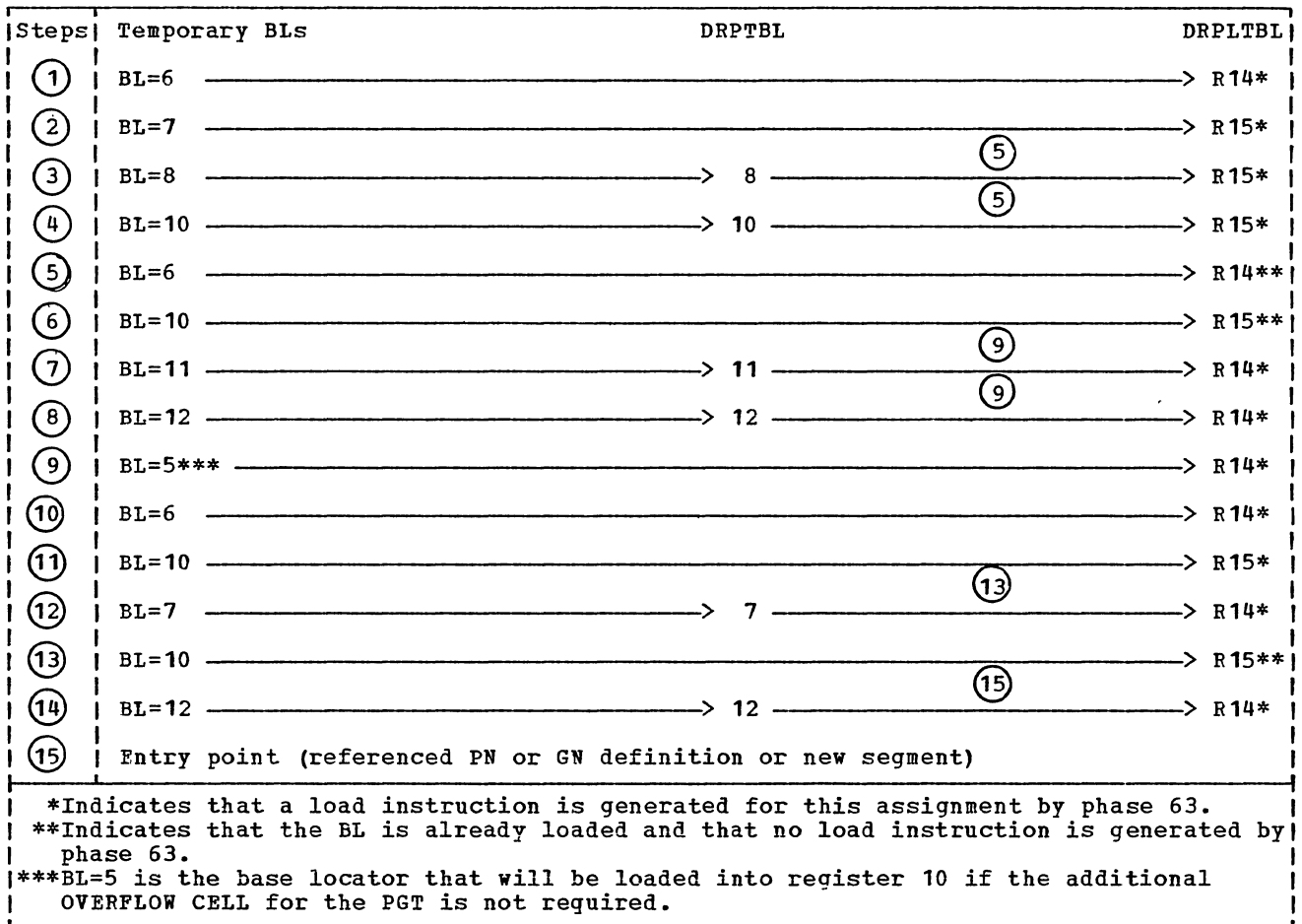


Figure 59. Optimizing Assignment of Registers 14 and 15

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 PNLBTBL and GNLBTBL tables. These tables are passed to phase 63, which generates the actual instructions necessary for establishing addressability.

To build the PNLBTBL and GNLBTBL 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 PNFWDBTB and GNFWDBTB tables for all PN's and GN's which 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 (PNFWDBTB and GNFWDBTB) 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 a PN or GN definition is encountered which has been entered into the PNFWDBTB or

GNFWDBTB table, the counter for that PN or GN is set to zero.

Since references to PN's and GN's which have not yet been defined may entail an additional instruction to load the Procedure Block address of a different block, the number of bytes represented by the counters in the PNFWDBTB and GNFWDBTB tables must be added to the displacement in ACCUMCTR to determine the current length of the 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, phase 62 enters the Procedure Block number of each referenced PN and GN definition into the PNLABTBL and GNLABTBL tables for use by phase 63.

Routine NOBLST 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. For this purpose it uses the codes listed in Figure 60, as well as Procedure A-text, and the PNLABTBL, GNLABTBL, PNFWDBTB, and GNFWDBTB tables.

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. Phase 62 then increments ACCUMCTR accordingly. The PNLABTBL and GNLABTBL tables, as well as the PNFWDBTB and GNFWDBTB tables and the PROCBL counter are used for this purpose.

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. Using the PROCBL counter, routine DEFILD11 enters the Procedure Block number of each PN or GN definition into the PNLABTBL and GNLABTBL tables, respectively.

When a branch is taken to a PN or GN within the Procedure Block whose address is already loaded into register 11, 4 bytes are added 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 routine ENTPT01 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 FWDBCTEL tables is greater than 4095 bytes. If it is not, then Procedure A-text processing continues. If it is, then the 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 to the number of bytes in ACCUMCTR minus 4096. If the count value is low, a new procedure block begins at this definition point. If it is high, the count field is zeroed and this definition point remains within the current block. Phase 62 then makes new calculations to determine the size of the Procedure Block.

CODE	MEANING/Procedure-Name Definition Status	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>1</sup> Add 4 to counters. Do not rewrite C001.	Fill in displacement using PNLBDTBL or GNLBDTBL table and Procedure base register element. <sup>1</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>2</sup> Generate LA instruction. Add Procedure base register element. <sup>1</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>1</sup>
	C Procedure-name is not yet defined (forward reference).	Enter procedure-name in PNFWDTBL or GNFWDTBL table. Resolve procedure-name definition status by end of Procedure Block.		

**Notes:**

<sup>1</sup>Procedure Base Register Element:

Bytes                    0                    1                    2-3

C8 = PN	Register number	PN/GN Number
CC = GN		

<sup>2</sup>Procedure Block Number Element:

Bytes                    0                    4

C4	Block Number
----	--------------

<sup>3</sup>These phase 50 Optimization Information elements (C0xx) are created by phase 50 from phase 40 Optimization Information elements (43xx).

Figure 60. Processing for Optimization Information Elements<sup>3</sup> (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 60.
C004	PERFORM exit.	Find all entries in the BLVNTBL for the VN whose reference follows this element. Enter current block number into these table entries.	Do not rewrite C004.	

**Notes:**

<sup>1</sup>Procedure Base Register Element:

Bytes	0	1	2-3
	C8 = PN CC = GN	Register number	PN/GN Number

<sup>2</sup>Procedure Block Number Element:

Bytes	0	4
	C4	Block Number

<sup>3</sup>These phase 50 Optimization Information elements (C0xx) are created by phase 50 from  
phase 40 Optimization Information elements (43xx).

Figure 60. Processing for Optimization Information Elements<sup>3</sup> (Part 2 of 3)

CODE	MEANING/Procedure-Name Definition Status	ACTION TAKEN BY								
		PHASE 62	PHASE 63	PHASE 64						
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>2</sup>						
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>2</sup>						
	C Procedure-name not yet defined.	Enter procedure-name into PNFWDDBT or GNFWDDBT; 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.						
<b>Notes:</b>										
<sup>2</sup> Procedure Block Number Element:										
<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;">4</td> </tr> <tr> <td></td> <td style="text-align: center;">C4</td> <td style="text-align: center;">Block Number</td> </tr> </table>					Bytes	0	4		C4	Block Number
Bytes	0	4								
	C4	Block Number								
<sup>3</sup> These phase 50 Optimization Information elements (C0xx) are created by phase 50 from phase 40 Optimization Information elements (43xx).										

Figure 60. Processing for Optimization Information Elements<sup>3</sup> (Part 3 of 3)

## PHASE 63

Phase 63 is the second of the three phases that make up the optimizer section of the compiler. Its principal function is to produce Procedure A1-text, which is written on file SYS002. 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 produce the optimized machine language program. Phase 63 also causes the Procedure Block assignments to be written on SYSLST (SYS006 for LVL option) if the LISTX, CLIST, or SYM options are 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 of priority if it is segmented.

The operations of Phase 63 are described in Diagram 8.

### 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 file SYS001. Otherwise, the initialization routine requests phase 00 to read the first Procedure A-text buffer from file SYS001.

### CONSTRUCTING PROCEDURE A1-TEXT

Phase 63 reads Procedure A-text from file SYS001 and writes Procedure A1-text on file SYS002.

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 C0 elements, macro-type instruction elements, and operation code elements, it branches to routines C0, MACRO, and FOURTY8, respectively.

For each of the other elements it uses the GETBTBL table to branch to the proper routine for specific processing of that element.

### PROCESSING PROGRAMS WITH ONE PROCEDURE BLOCK

In programs which do not exceed one Procedure Block in length and which have no Report Writer or Declaratives Section, and in which segmentation does not occur, the Procedure Block address is loaded into register 11 only when the ENTRY macro (4404) and/or START macro (4420) of Procedure A-text is read.

### PROCESSING FOR BRANCH INSTRUCTIONS

Routine BRANCH processes the branch element following the PN or GN reference. It uses the SAVETBL and either the PNLABTBL or GNLABTBL table to determine whether the PN or GN referenced by this branch is defined in the Procedure Block that is currently loaded in register 11. If the PN or GN is defined outside of the Procedure Block currently loaded, a Procedure A1-text element is generated to load register 11 with the address of the Procedure Block which contains the PN or GN definition. Routine BRANCH then generates an RX-type Procedure A1-text branch element instead of the RR-type of Procedure-A text 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 60 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 ORG macro 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 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 element is found in Procedure A-text, routine GET branches to routine ADREF for processing.

#### PROCESSING FOR ADDRESS INCREMENT (80) ELEMENTS

When address increment elements occur, they follow Address reference elements and indicate that an additional displacement

value is to be added to the value indicated by the Address reference element to address a data-name.

When Routine GET finds an Address increment 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 element is less than 4096 bytes. If the sum is less, then routine ADINCR adds a byte containing X'00' to the Address reference element and writes the element in Procedure A1-text.

If routine ADINCR 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 LA instructions are to be generated using either register 14 or register 15, respectively, to the Address increment 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. Routine ADREF (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 it does, 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.

#### 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 GNLABTBL by Phase 63 at each GNDEF 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" above.

#### PROCESSING IN A SEGMENTED PROGRAM

When a program is not segmented, phase 63 reads Procedure A-text from file SYS001 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.

Routine PHAS63 first determines whether the program is segmented by checking SEGLMT in COMMON. If SEGLMT does not contain X'FF', the routine relocates SEGTLB 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.

When routine MACRO comes to the end of a section, it branches to routine SEGBRK, 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 zeros out LOCCTR and ACMCTR and points to the next segment of next highest priority; the routine also calls the routine SAVTCTBL to save the address of the end of the root segment, which is either the address of INIT2 or, if COUNT is in effect, the address of the COUNT table. If it is not the end of the segment, the routine points to the next section of the same priority. When the end of the SEGTLB table is reached, control passes to routine EOF.

#### 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, GNLBDTBL, 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

Phase 64 is the third of the three phases which make up the optimizer section of the compiler. It completes the necessary processing to produce the machine-language program. The major functions of phase 64 are:

- Processing Data A-Text and completing the RLDTBL table.
- Processing Procedure A1-text, and entering displacements into the instructions generated in phase 63.
- Writing object text and RLD-text from the RLDTBL table.
- Writing object text from Procedure A1-text
- Writing object text for the INIT2, INIT3, and INIT1 routines of the object program, in that order.
- Building the QTBL and ERRtbl tables.
- Processing DEF-text, E-text, and REF-text.

OUTPUT OF PHASE 64

The output of phase 64 depends on the options specified by the user. The LISTX, CLIST, LINK (or CATAL), and DECK options are processed by phase 64 in the same way as they are processed by phase 60. For details, see "Output of Phase 60" in the chapter "Phase 60."

If the SXREF, XREF, VERBREF, or VERBSUM option is in effect, Procedure A1-text is used to create REF-text and to write it on file SYS004. This text, containing an element for every data-name, file-name, and procedure-name in the program, is used by phase 61 to produce a cross-reference listing. DEF-text, which is also produced in response to the specification of the SXREF, XREF, VERBREF, or VERBSUM option, is read by phase 64 from file SYS004 and passed to phase 61 on file SYS001.

COMPLETING THE RLDTBL TABLE

Phase 64 reads Data A-text from file SYS004 before it reads Procedure A1-text

from file SYS002. 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. Before phase 64 reads Procedure A1-text, therefore, it must complete all RLD entries for the root segment.

The primary function of Data A-text processing is to place values in fields in the data area or in the global tables in the object module. Each element causes either the generation of an object text element or the creation of an entry in the RLDTBL or QTBL table. Routine RLDSORT is used to make entries in the RLDTBL table in sorted order. Some RLDTBL table entries are later written out as relocation dictionary (RLD-text) entries for the data area and as object text. Other entries (for the global tables) will be written out as object text only (these will be relocated by the object program).

File SYS004, from which Data A-text is read, also contains E-text, generated by phases 10 through 63, and DEF-text for the cross-reference listing if the SXREF, XREF, VERBREF, or VERBSUM option was specified. Figure 57 in the chapter "Phase 60" illustrates the contents of this file when it is read by phase 64. E-text and DEF-text are processed by phase 64 in the same way as they are processed by phase 60. Figure 58 in the chapter "Phase 60" describes how each type of element is processed.

After end of file on SYS004, Procedure A1-text is processed.

COMPLETING THE MACHINE LANGUAGE PROGRAM

Routine SE6000 reads Procedure A1-Text from file SYS002. Since Procedure A1-Text has been written by phase 63 in order of ascending priority with the root segment first, routine SE6000 reads the text sequentially for segmented programs, as well as for non-segmented programs.

The special processing done by phase 64 for Procedure A1-text elements is described in Figure 61 "Processing of Procedure A1-text." All other Procedure A1-text elements are processed by phase 64 in the same way that phase 60 processes Procedure A-text elements. See "Procedure A-Text Processing" and Figure 56 "Processing of Procedure A-Text" in the chapter "Phase 60."

Code and Type	Action Taken
78 address reference	If the i field contains X'03' the BL, BLL, SBL, or SBS indicated is already loaded. Process in the same way as phase 60 does. Otherwise, save contents of print buffers and generate load of register 14 or register 15 with the BL, BLL, SBL, or SBS indicated by the high-order bit of the i field. If the bit is on, use register 15; if it is off, use register 14. Restore buffers and complete processing in the same way as phase 60 does.
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."
EC 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, with Procedure Block'
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 set in COMMON by 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; branch to routine for processing dictionary pointer.
*This table describes only those elements which are unique to Procedure A1-text. All other elements are identical with their counterparts in Procedure A-text, and are processed by phase 64 in the same way as they are processed by phase 60. See "Processing of Procedure A-text" in the chapter "Phase 60."	

Figure 61. Processing of Procedure A1-Text\*

INITIALIZATION ROUTINES

After Procedure A1-text has been processed, the initialization coding is generated. Routines GINIT1, GINIT2, and GINIT3 generate the code for INIT1, INIT2, and INIT3, respectively. The three initialization routines, in the order in which phase 64 writes them, are INIT2, INIT3, and INIT1. All three are resident in the root segment if the program is segmented.

INIT1 sets up address constants for the program's TGT, PGT, and first executable instruction, and for the three initialization routines. It then transfers control to INIT2 if the program is a subprogram, or to INIT3 if it is not.

INIT2 is executed if the program is a subprogram or is entered at a secondary entry point. It establishes standard subroutine linkage. Control then passes to INIT3 which sets up base registers, relocates addresses, and transfers control to the proper point in the program to begin execution. (For a more complete description of these three routines,

including the generated code, see "Appendix B. Object Module.")

RLDTBL TABLE PROCESSING

After the initialization routines are generated (after Procedure Division or root segment processing), the RLDTBL table is processed. 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. No RLD-text is required for these items, because the addresses are relocated during program execution by routine INIT3. However, in a nonsegmented program, 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.

PHASE 61

The function of phase 61 is to produce a cross-reference listing on SYSLST (or SYS006 for LVL option). The phase is given control only if the SXREF, VERBREF, VERBSUM, or XREF compiler option was specified by the user. The transfer of control is described under "Processing Between Phases" in the chapter "Phase 00."

Phase 61 tests the PHZSW2 byte in COMMON to determine whether the option specified is SXREF. If SXREF is in effect, phase 61 generates an alphabetically ordered cross-reference listing. Phase 61 tests the PHZSW4 byte in COMMON to determine whether the option specified is VERBREF or VERBSUM. If VERBREF is in effect, phase 61 produces the verb cross-reference listing. If VERBSUM is in effect, phase 61 produces only the verb summary listing. Otherwise, XREF is in effect and a cross-reference listing ordered by source statement sequence is generated.

Phase 61 performs the following operations:

- Reads DEF-text into storage until either storage is filled or end-of-file is reached. It 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 three operations constitute the fundamental cycle which 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 (VERBREF 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 61 uses the GETALL routine in TAMER 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. It does not, however, use TAMER routines to access these tables. phase 61, therefore, 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. An algorithm is used for dividing the area into two parts, one for the OFLOTBL and one for the DATATBL.

The DEF-text is read from file SYS001 where it was written by phase 60, 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 file SYS001 is read into storage and the contents of each element are moved into the corresponding area in a DATA record. This is repeated until all assigned DATA records (for which there was space) are filled with DEF-text. One DATA record is created from each DEF-text element.

The REF-text is read from file SYS003 or, if OPT is in effect, from file SYS001. There is one element of REF-text for each time a name is referred to in the source program. The text on file SYS003 or SYS004 is read, one element at a time, until end-of-file is reached.

In the REF-text, 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 11. 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 happened to have the same bit configuration.

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 which was chained to it when the first overflow record was needed. 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 space is available for the OVERFLOW record; the DATA record containing the last DEF-text element read is taken out of the DATATBL and the space is assigned to the OFLOTBL. The space is divided 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. At end-of-file, the REF-text file is closed.

At the end of the cycle, each DATA record and its associated OVERFLOW records are printed on SYSLST (or SYS006 for LVL option).

If this is the last or only cycle, processing is completed when all names, defining card numbers, and referencing card numbers in main storage have been printed. If this is not the last cycle, DEF-text is again read into main storage. (If it was necessary to split one or more DATA records into OVERFLOW records in the preceding cycle, the DEF-text file must be rewound, and the DATATBL and OFLOTBL set back to their original lengths. REF-text must also be reread. 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 routine GETALL in Phase 00. An algorithm is used for dividing the space among the three tables.

The DEF-text is read from file SYS001. The text is read into storage and the contents of each element are moved into the corresponding area in a DATA record. This is repeated until all assigned DATA records (for which there was space) are filled with DEF-text. 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 file SYS003, or, if OPT was specified, from file SYS004. The text on file SYS003 or SYS004 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 space is available for an OVERFLOW record, the DATA record which 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 file is closed.

At the end of the cycle, each DATA record and its associated OVERFLOW records are printed in alphabetical order on SYSLST (or SYS006 for LVL option). 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 file, the file must be rewound.) 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.

PHASE 70

Phase 70 (ILACBL70) generates all the compiler diagnostics for source program errors. Its input consists of E-text from phases 01 through 51 that is either in storage or on SYS003. If no output from phases 60, 62, 63, or 64 has been requested or if a message of sufficient severity has been generated and the SUPMAP option is in effect, phases 60, 62, 63 and 64 text processing is bypassed, and phase 70 reads its input from SYS004. Its output consists of completed messages, which are written on SYSLST (or SYS006 for IVL option).

INPUT FROM PRIOR PHASES

Phases 01 through 51 produce E-text in the same manner. Whenever a processing routine detects a source program error, it writes out message definition elements of E-text described in "Section 5. Data Areas."

If parameters are associated with the error message, the phase sets up a message parameter entry immediately after the error entry.

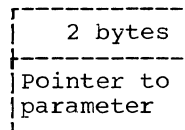
Phase 01 writes E-text for BASIS and COPY statements on SYS003. Phase 10 writes E-text intermixed with Data A-text on SYS003. Phase 21 reads E-text, interspersed with other texts, from SYS003. It writes the E-text back onto SYS004 without change, along with Data A-text and its own E-text. Phases 12 and 11 writes the E-text produced during P0-text processing on SYS002 intermixed with the P0-text. From then until phase 51, E-text is added to the Procedure IC-text stream as errors are encountered. Phase 51 isolates this E-text and writes it, together with its own, on SYS004. Phase 60 or phase 64 encounters E-text on SYS004 during its operations. To avoid extra input/output operations, phase 60 or phase 64 attempts to save the E-text in a storage area, the ERRTBL table. The ERRTBL table, however, is of a fixed size. If all the E-text cannot be saved in ERRTBL phase 60 or phase 64 writes the E-text on SYS003 for phase 70 to read. (ERRTBL is of a fixed size to allow all available space to be assigned to the RLDTBL during phase 60 processing.)

PHASE 70 ERROR PROCESSING

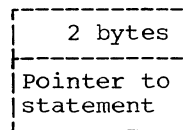
Upon receiving control from phase 00, phase 70 uses the PARTBL, EACTBL, and PHxERR tables, along with E-text, to construct error messages which it then writes out.

THE PARTBL AND EACTBL TABLES

The PARTBL table is a fixed table assembled as part of the phase and not handled by TAMER routines. It contains pointers to all possible error message parameters (COBOL words, verbs, operations, etc.) that are not programmer-supplied names. The pointer is a displacement from the beginning of all parameters to the parameter for that entry. Its entries are of the form:



The EACTBL table is a fixed table assembled as part of the phase and not handled by the TAMER routines. It contains pointers to error statements which describe what compiler action was taken because of the error; for example, "STATEMENT ACCEPTED AS WRITTEN." Its entries are of the following form:



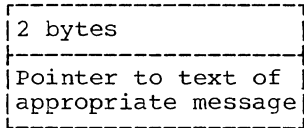
The pointer is a displacement from the start of the list of actions to the start of the action for this entry.

THE PHxERR TABLE

The PHxERR tables are fixed tables assembled as part of the phase and are not handled by TAMER routines. For each message, an entry is made in the appropriate PHxERR table, where x has the following values:

- 0 = Phase 00
- 1 = Phase 01 (BASIS/COPY), 10, 11, or 12
- 2 = Phase 20, 21, 22, or 25
- 3 = Phase 30
- 4 = Phase 40
- 5 = Phase 50 or 51
- 6 = Phase 60, 62, 63, 64, or 65

Each entry is of the following form:



The pointer is of the form of a displacement of the specific message from the start of messages for that phase.

The message text is found in phase 70 starting at label TX1000.

#### GENERATING MESSAGES

The XNORML routine scans each E-text item in turn. It first moves, into work area XU6REC, the card number, the message and phase numbers, and the severity code listed in the entry as follows:

bnnnnnbbILApxxxI-s

where:

- b indicates a blank
- nnnnn is the compiler-generated number of the statement containing the error
- p indicates the phase in which the error occurred, where:

- 1 = Phase 01, 10, 11, or 12
- 2 = Phase 20, 21, 22, or 25
- 3 = Phase 30
- 4 = Phase 40
- 5 = Phase 50 or 51
- 6 = Phase 60, 62, 63, 64, or 65

xxx is the number of the message

- s is the severity code, as follows:
  - W = warning
  - C = conditional
  - E = error
  - D = disaster

Routine XNORML 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 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 label EACT00. Routine XNORML moves the error action message into the work area immediately following the text for the message. Routine XPUT then writes the message on SYSLSY (or SYS006 for LVL option).

#### ERROR MESSAGE LISTING

At phase 70 initialization, control transfers to a string routine to generate a comprehensive listing of all compiler messages if the PROGRAM-ID is ERRMSG. The SEVTBL table and SC MACRO instruction are used for this purpose. A description of the listing is given in "Section 6: Diagnostic Aids."

## PHASE 80

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. The transfer of control is described under "Processing Between Phases" in the chapter "Phase 00."

When phase 01 determines which level of flagging has been specified (A = low; B = low intermediate; C = high intermediate; D = full standard), the level is stored in the System Communication Region. Phase 80 picks up the level from there via the COMRG macro; the LINECOUNT for the listing output is also picked up.

### Input

Input to phase 80 is the COBOL source program listing and other data on the SYS006 utility data set.

When FIPS processing has been requested, the source program is written on SYS006 by phase 01 if no Lister options are in effect or by phases 10, 12, and 11.

### Output

The output of Phase 80 is written on SYSLST. It consists of the COBOL source program listing, flagged according to the specified level of the Federal Information Processing Standard and other data written on SYS006 during compilation. However, if the LST option is in effect, the listing of the source program will be suppressed to avoid duplicating the Lister option listing; FIPS messages will be printed.

### Processing

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 SYSLST.

### 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 ILACBL80 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. The routine formats the message 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 SYSLST. 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

This chapter contains flowcharts of all the phases of the compiler. There is an overall flowchart for each phase, followed in most cases by more detailed flowcharts of the major routines in the phase. Also provided is a set of flowcharts for a typical Report Writer subprogram generated by the compiler.

The contents of this chapter are:

1. Explanation of flowchart symbols
2. Phase flowcharts (Charts AA through TA)
3. Report Writer Subprogram flowcharts (Charts UA through UT)

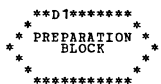
Explanation of Flowchart Symbols

FUNCTIONAL SYMBOLS

```
*****A1*****
*          *
* PROCESSING *
*   BLOCK   *
*          *
*****
```



```
*****C1*****
*ENTRY, WAIT, OR*
*TERMINAL BLOCK*
*****
```



```
*****E1*****
*INPUT/OUTPUT  *
*   BLOCK      *
*****
```

```
*****P1*****
*-----*
* SUBROUTINE  *
*   BLOCK     *
*****
```

```
*****G1*****
** PREDEFINED **
**   PROCESS   **
**           **
*****
```

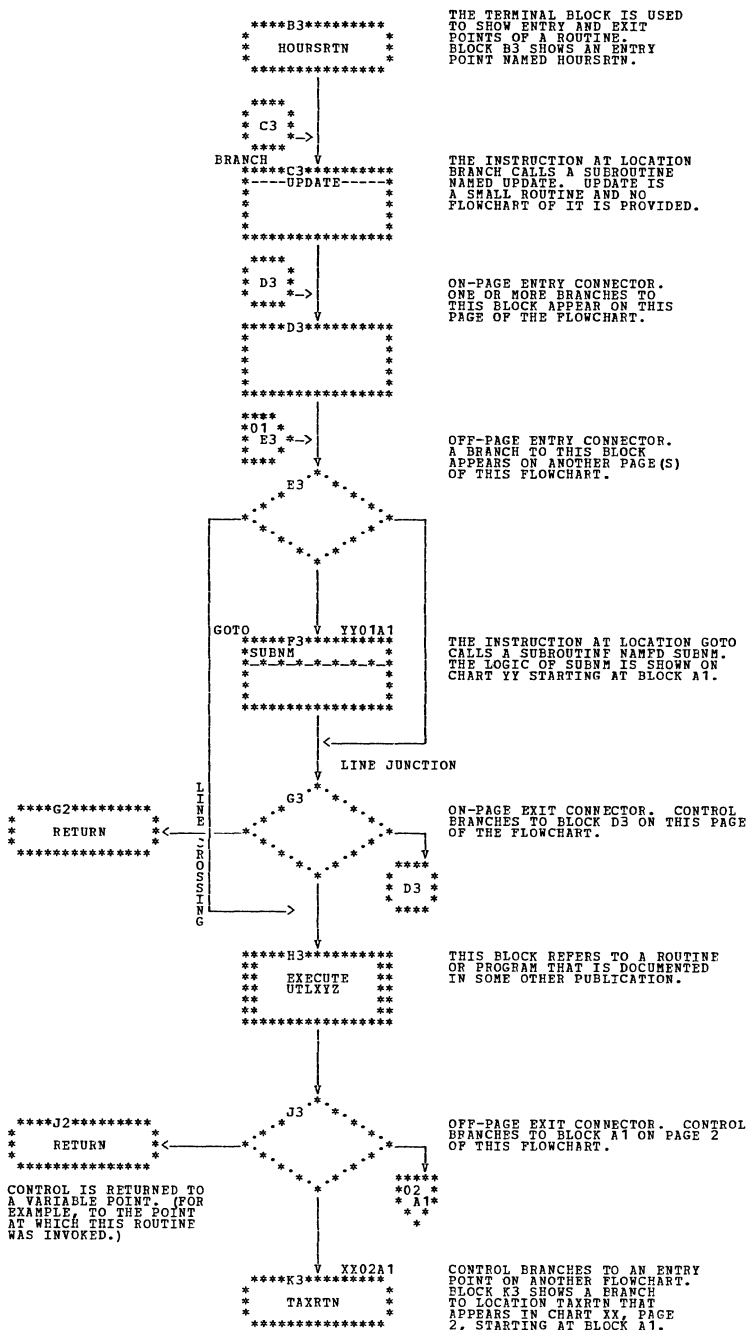
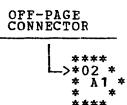
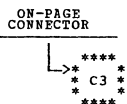


Chart AA. Phase 00 (ILACBL00): Overall Logic (Part 1 of 7)

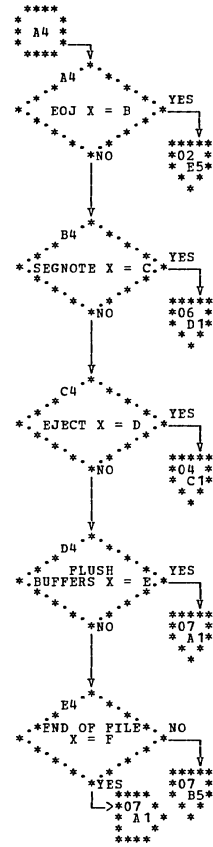
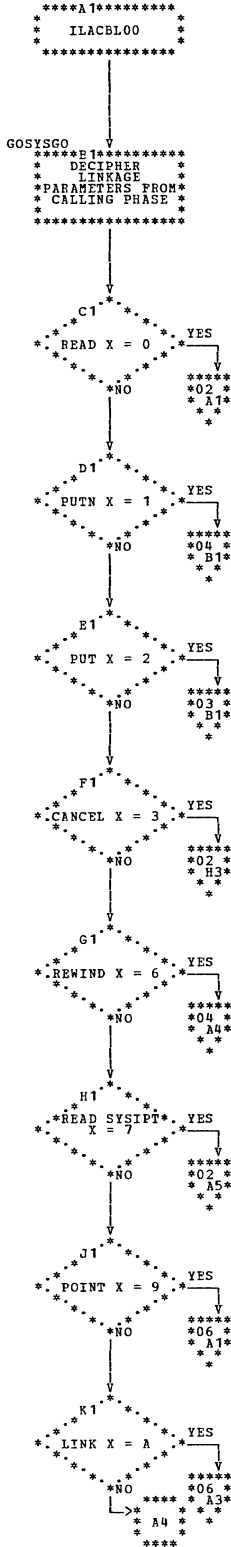


Chart AA. Phase 00: Overall Logic (Part 2 of 7)

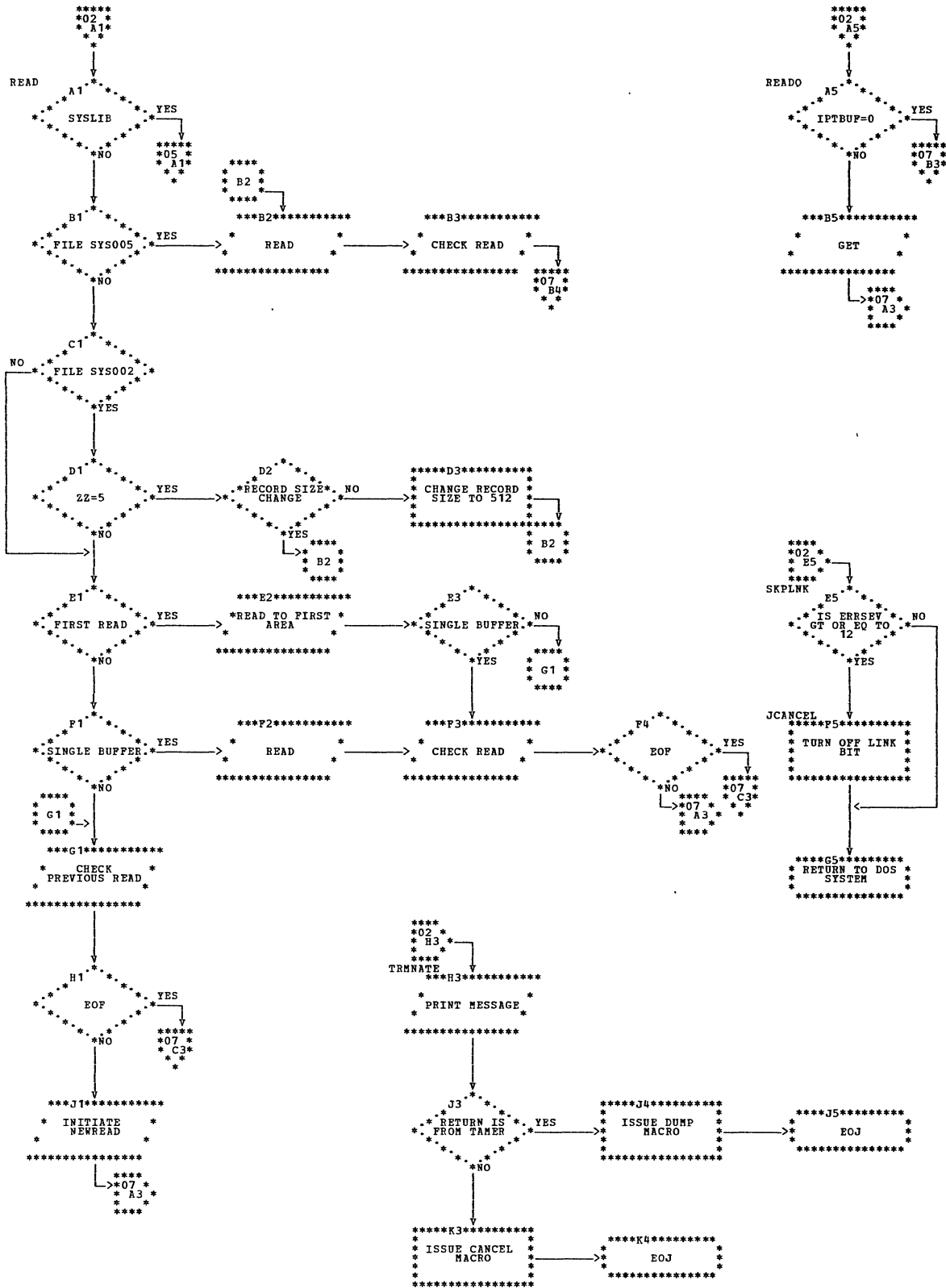


Chart AA. Phase 00: Overall Logic (Part 3 of 7)

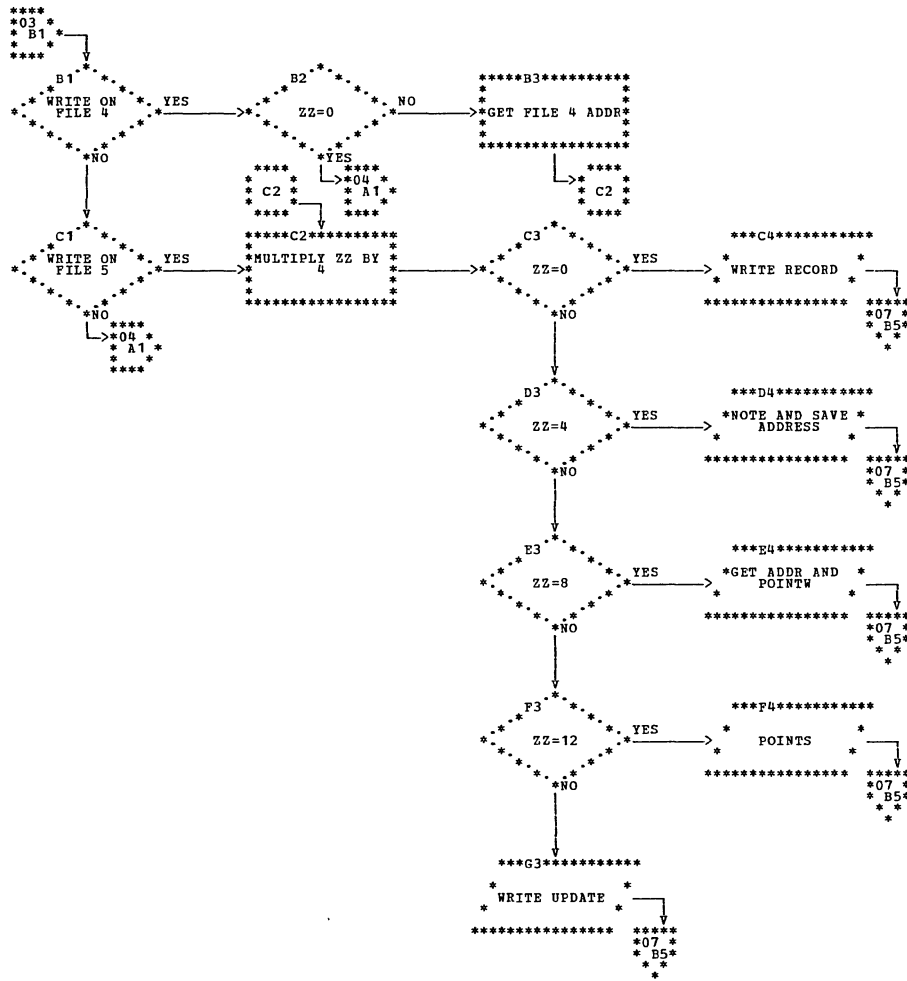


Chart AA. Phase 00: Overall Logic (Part 4 of 7)

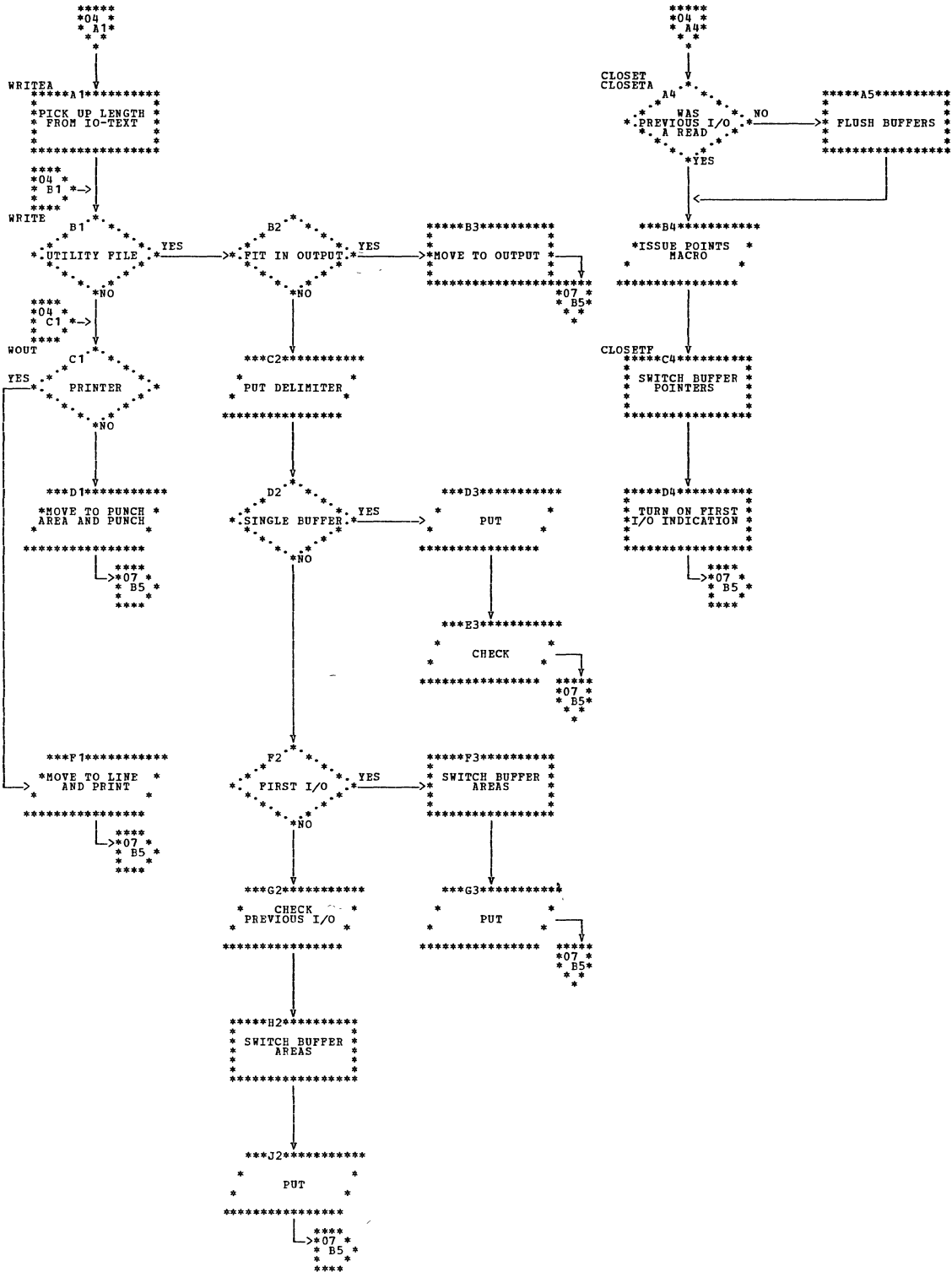


Chart AA. Phase 00: Overall Logic (Part 5 of 7)

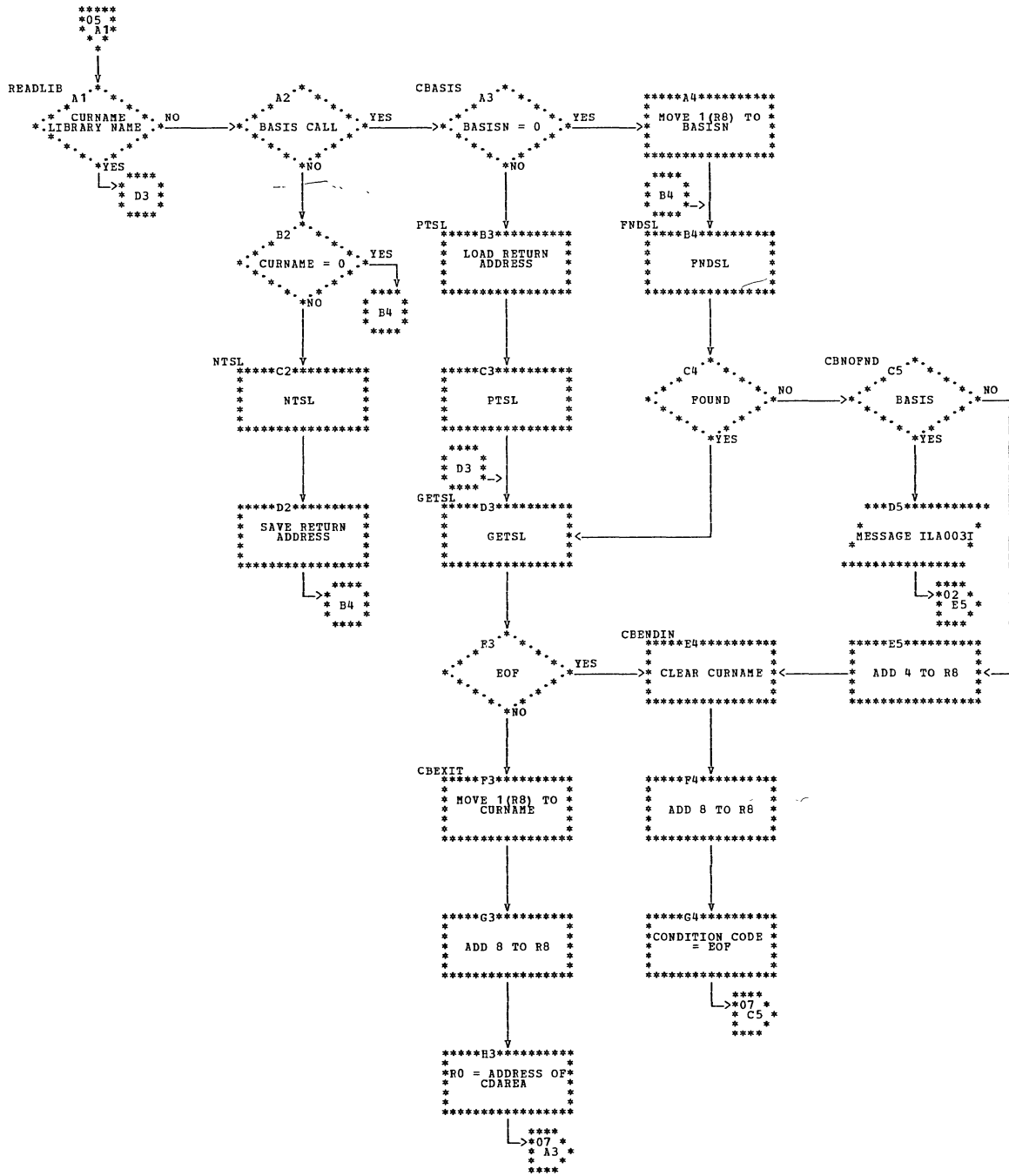


Chart AA. Phase 00: Overall Logic (Part 6 of 7)

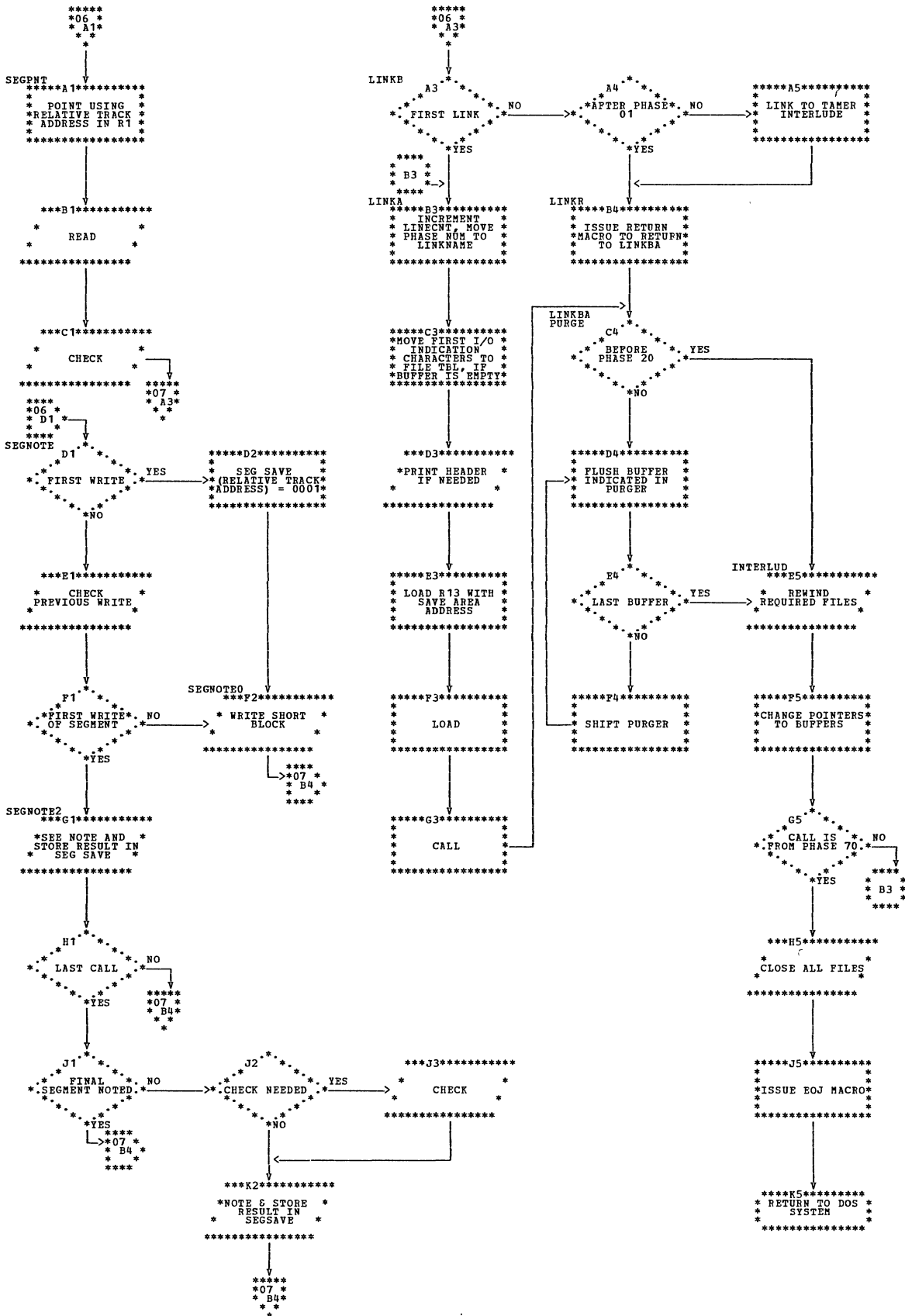




Chart AA. Phase 00: Overall Logic (Part 7 of 7)

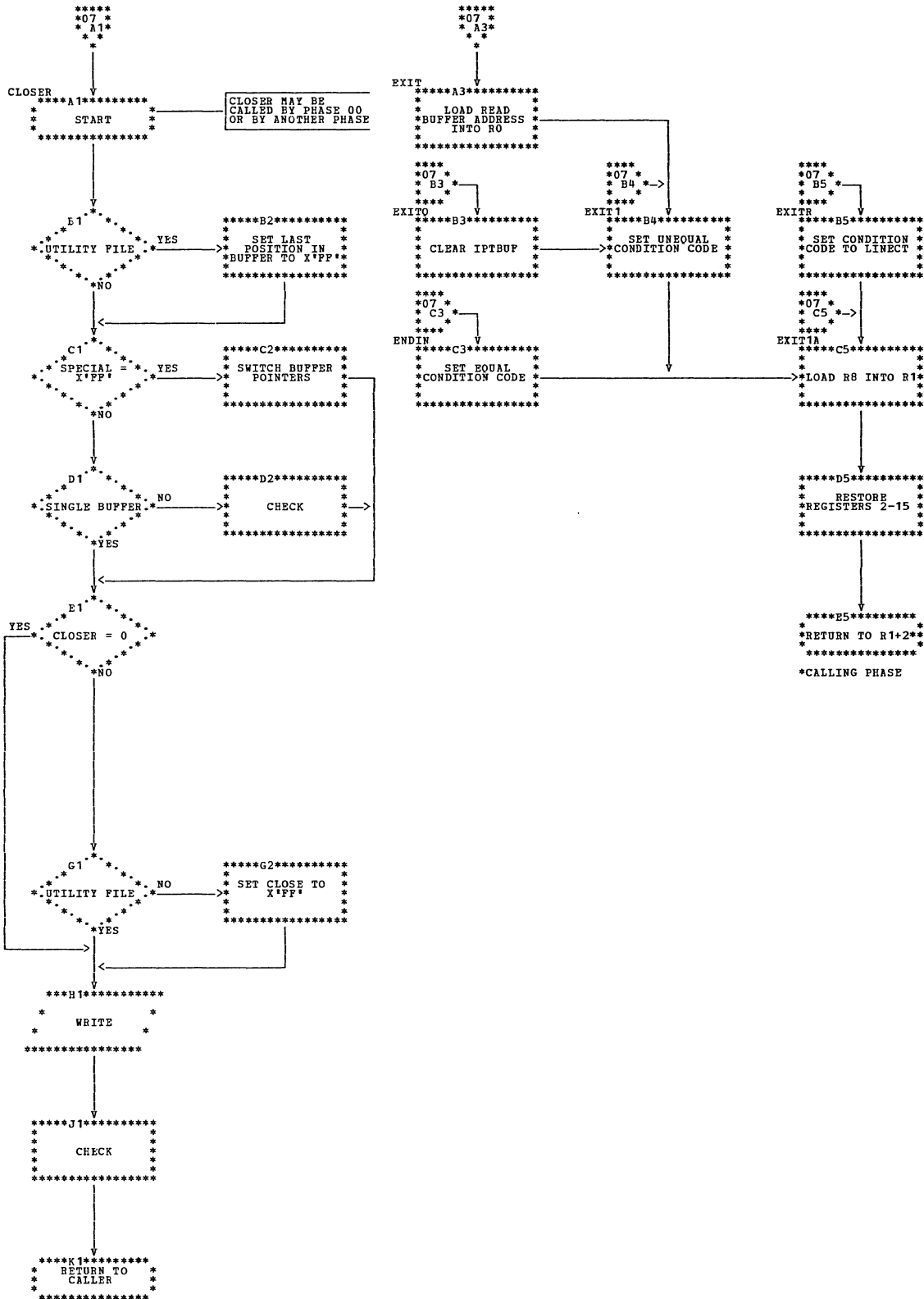


Chart BA. Phase 01 (ILACBL01): Overall Logic (Part 1 of 2)

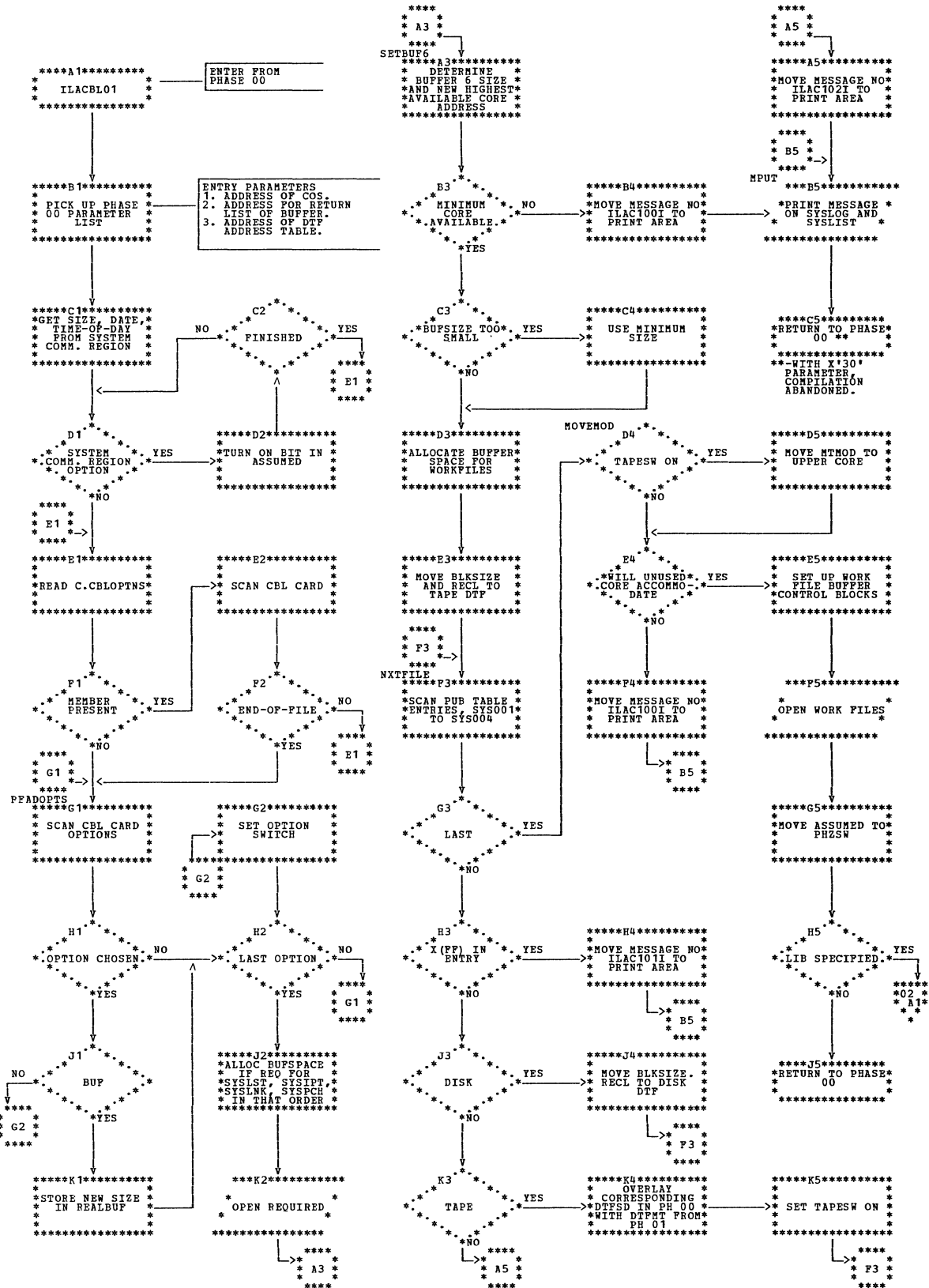


Chart BA. Phase 01 (ILACBI01): Overall Logic (Part 2 of 2) COPY/BASIS Functions

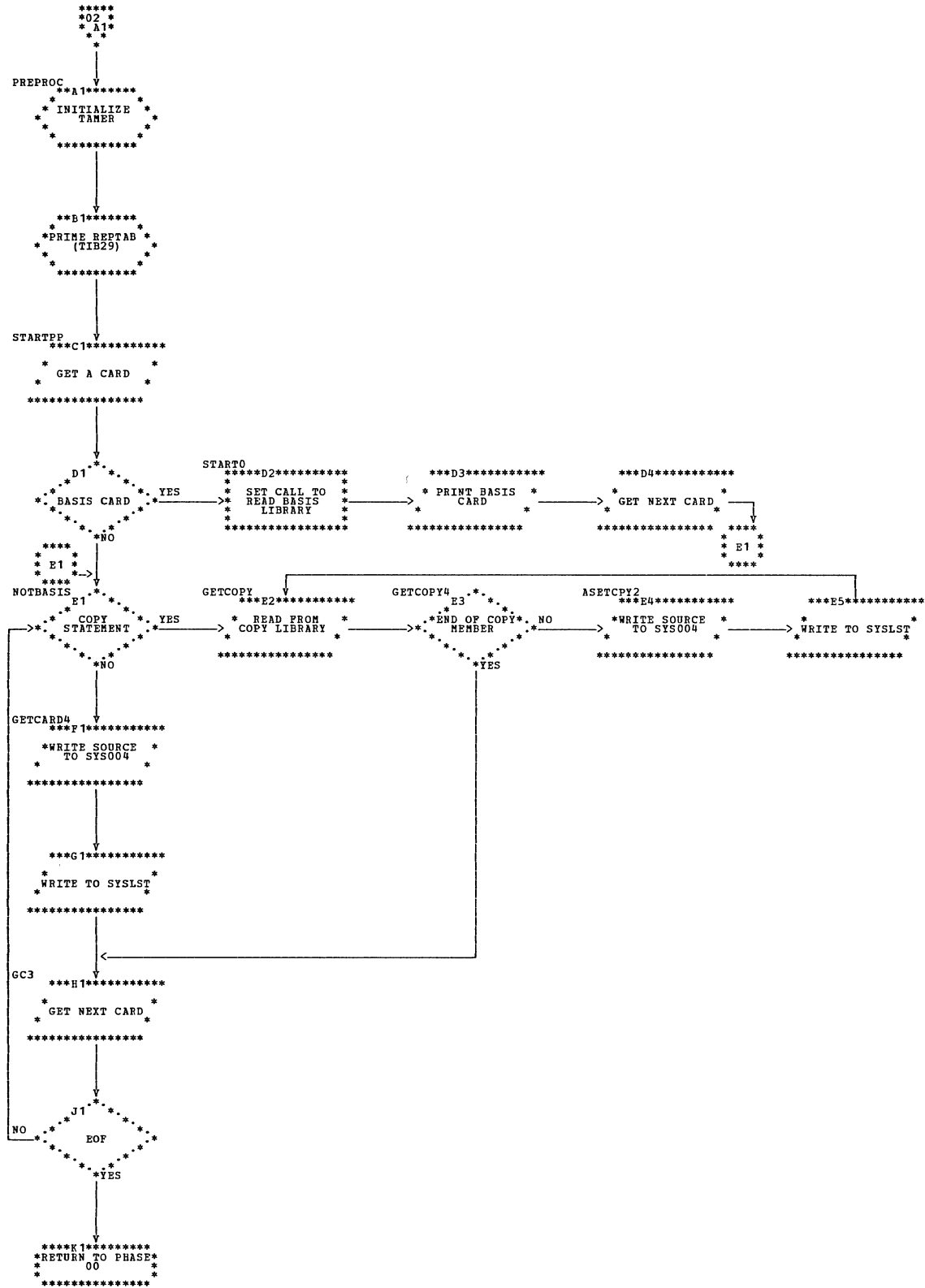


Chart CA. Phase 10 (ILACBL10): Overall Logic

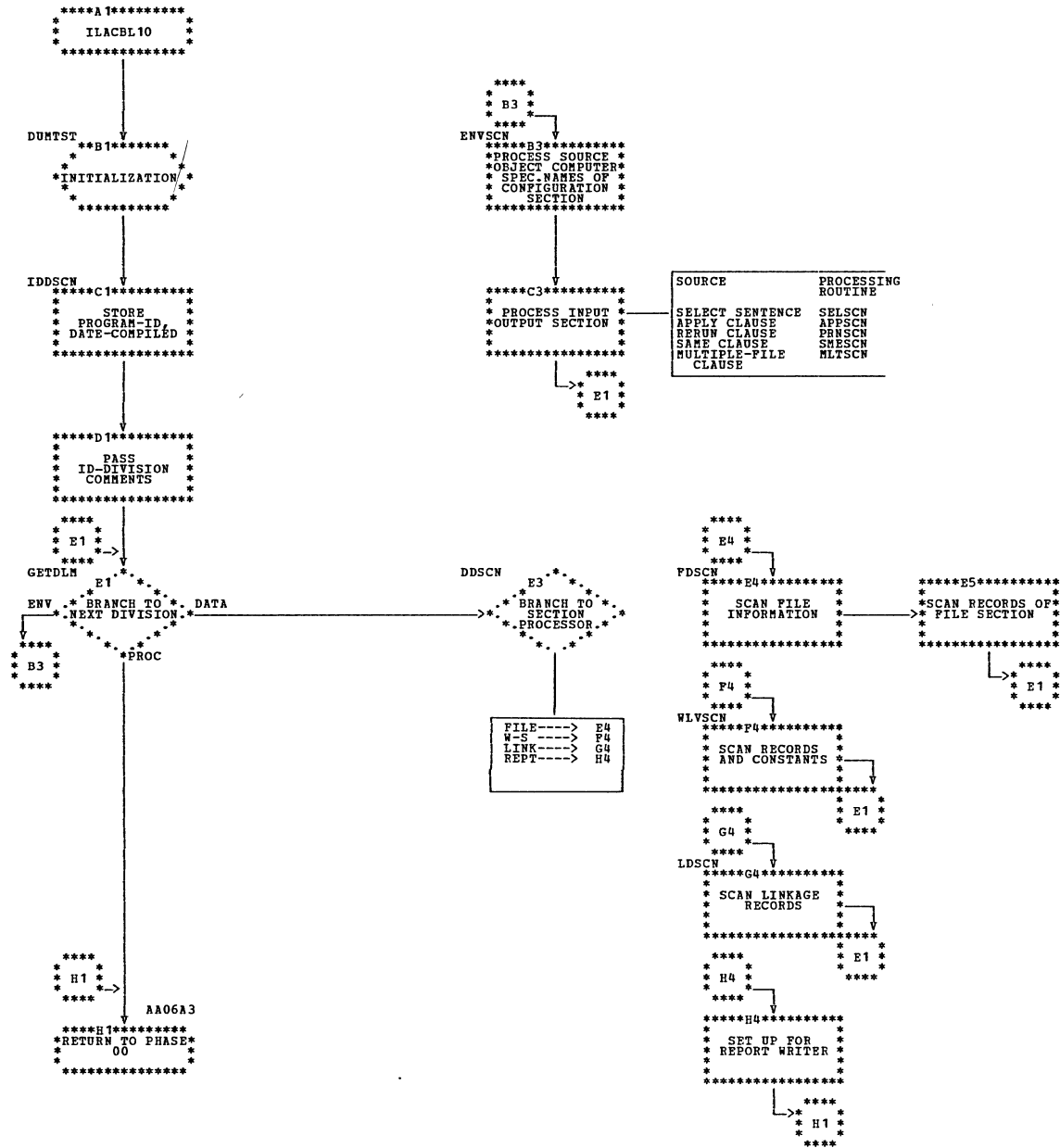


Chart DA. Phase 11 (ILACBL11): Overall Logic

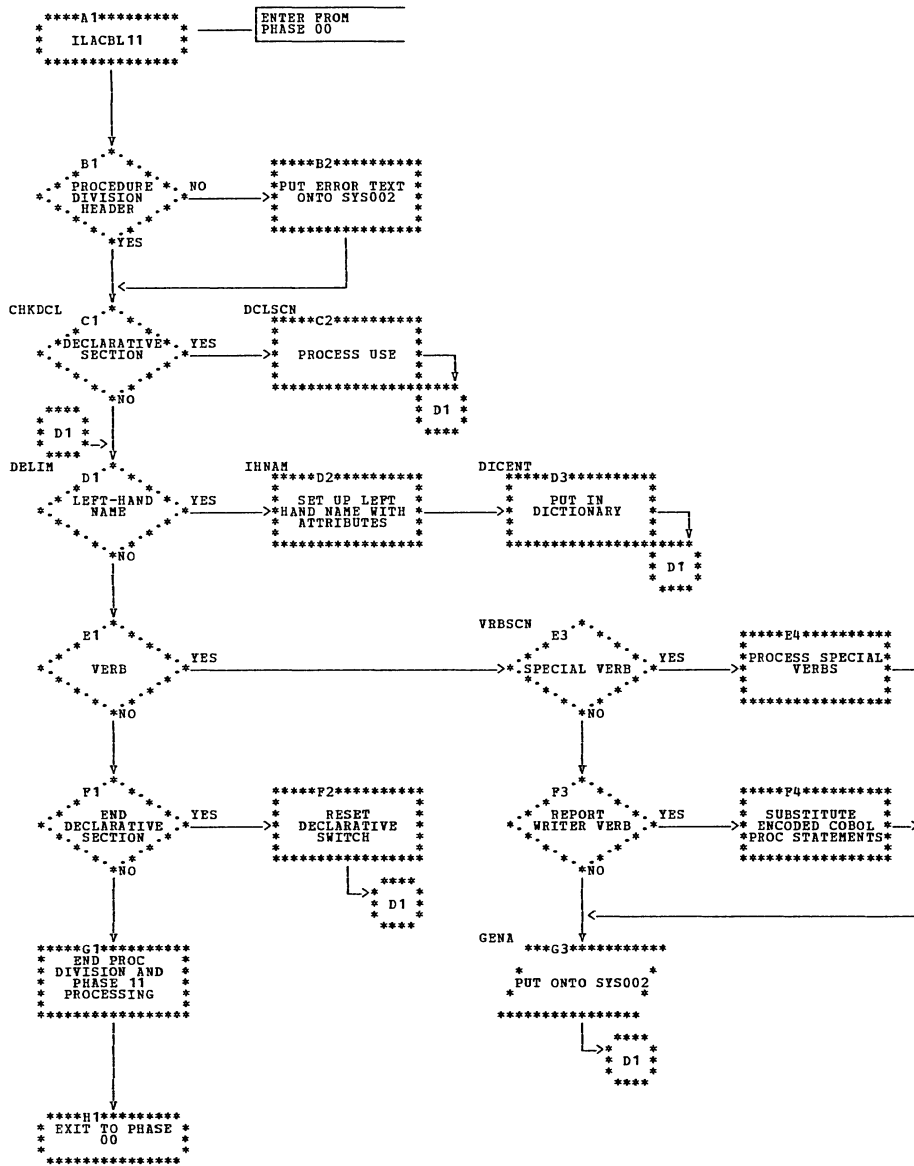
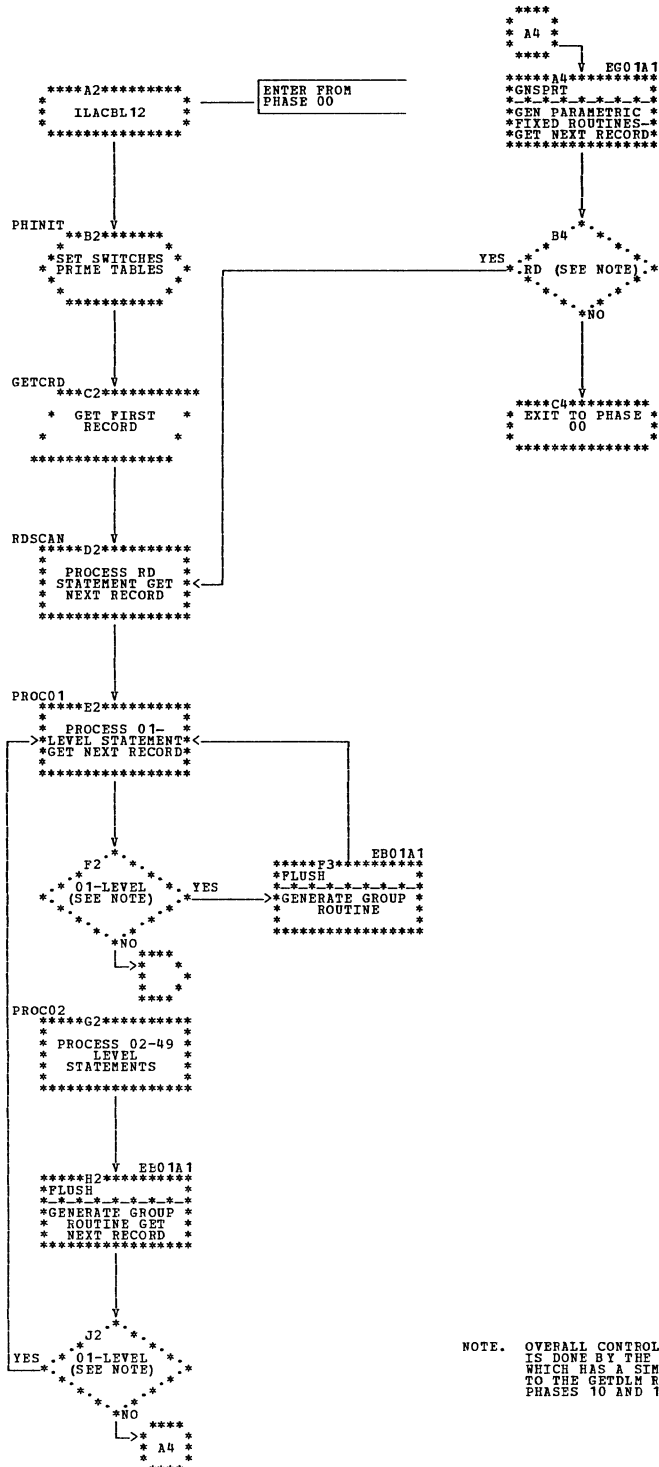


Chart EA. Phase 12 (ILACBL12): Overall Logic



NOTE. OVERALL CONTROL OR PROCESSING IS DONE BY THE GETDLM ROUTINE, WHICH HAS A SIMILAR FUNCTION TO THE GETDLM ROUTINE IN PHASES 10 AND 11.

Chart EB. Phase 12: FLUSH Routine

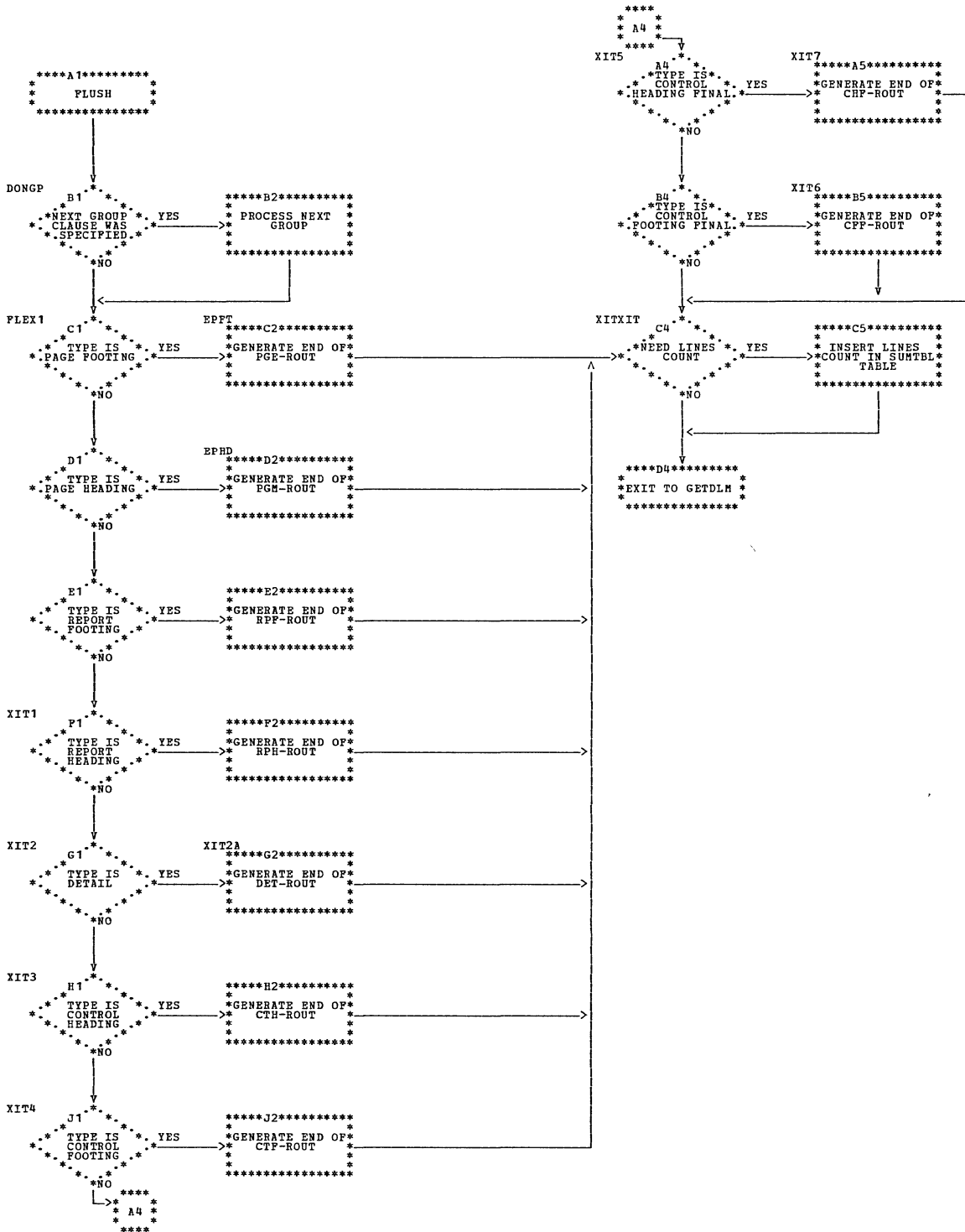


Chart FA. Phase 20 (ILACBL20): Overall Logic

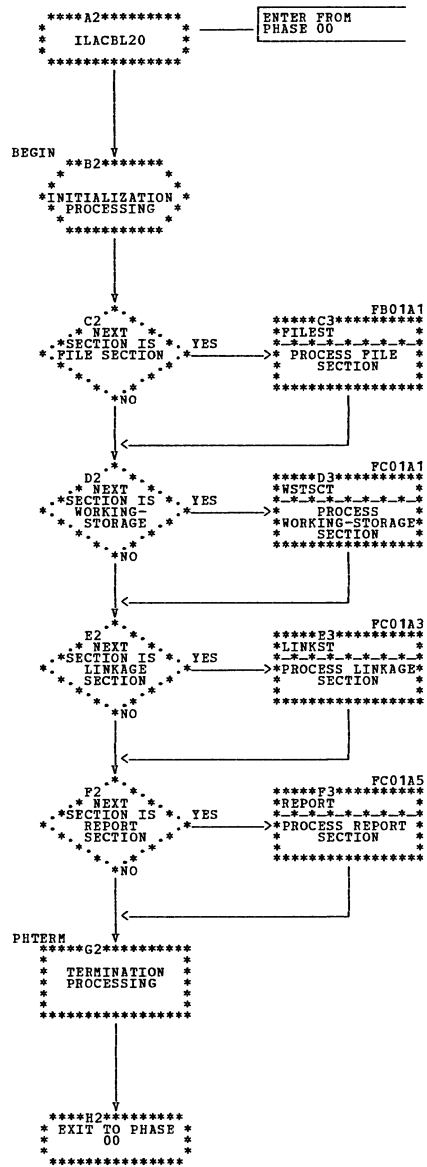






Chart FC. Phase 20: WSTSCT, LINKST, and REPORT Routines

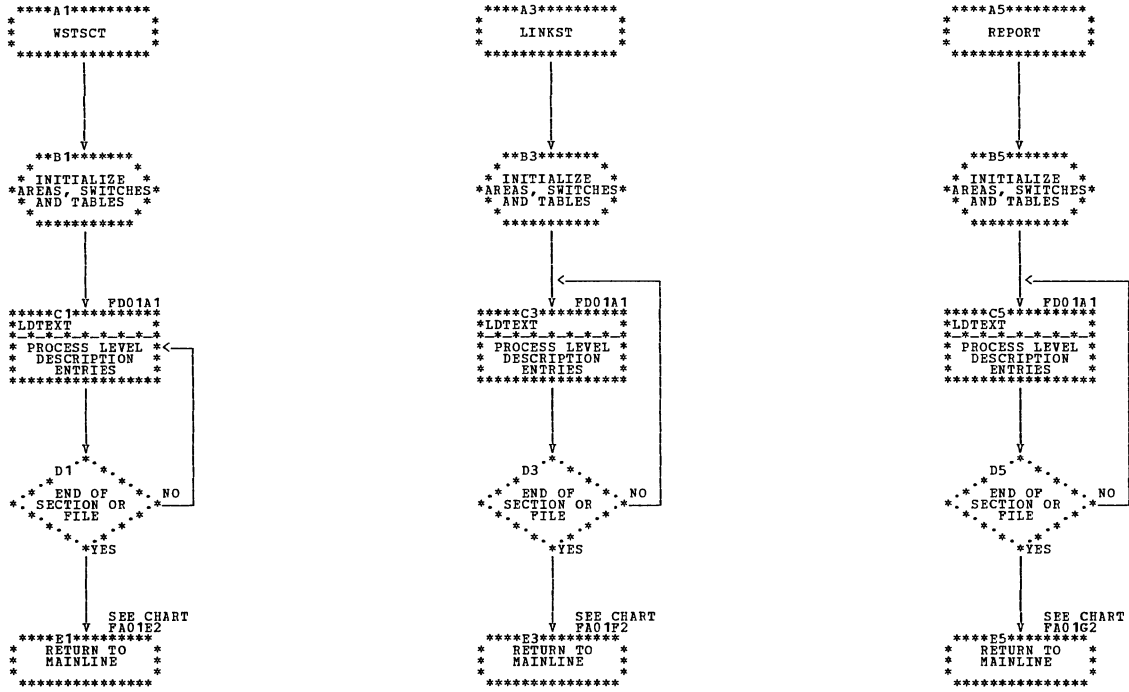


Chart FD. Phase 20: LDTEXT Routine

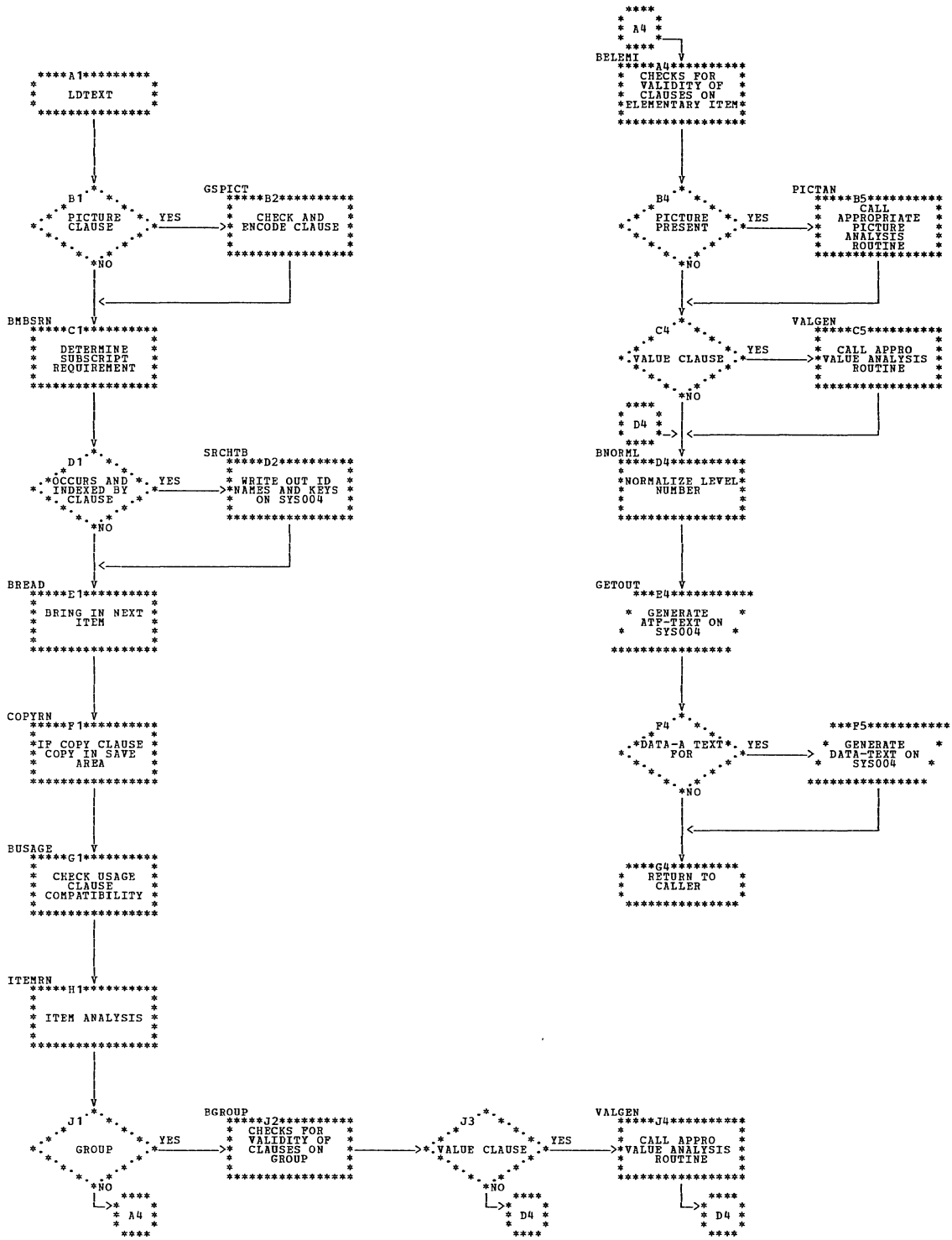


Chart GA. Phase 21 (ILACBL21): Overall Logic

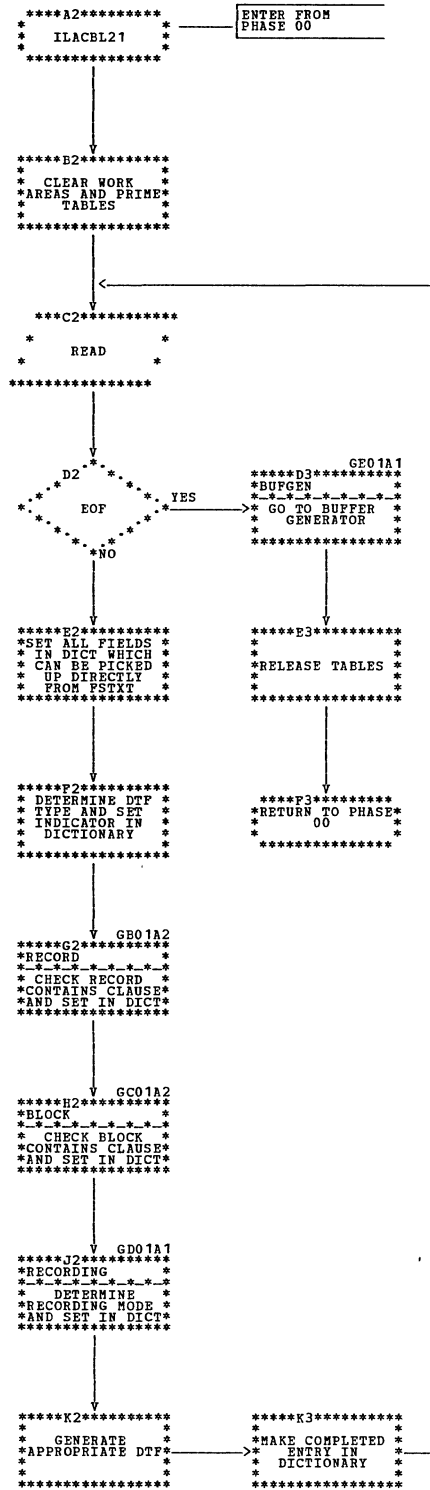


Chart GB. Phase 21: RECORD CONTAINS Clause Processing

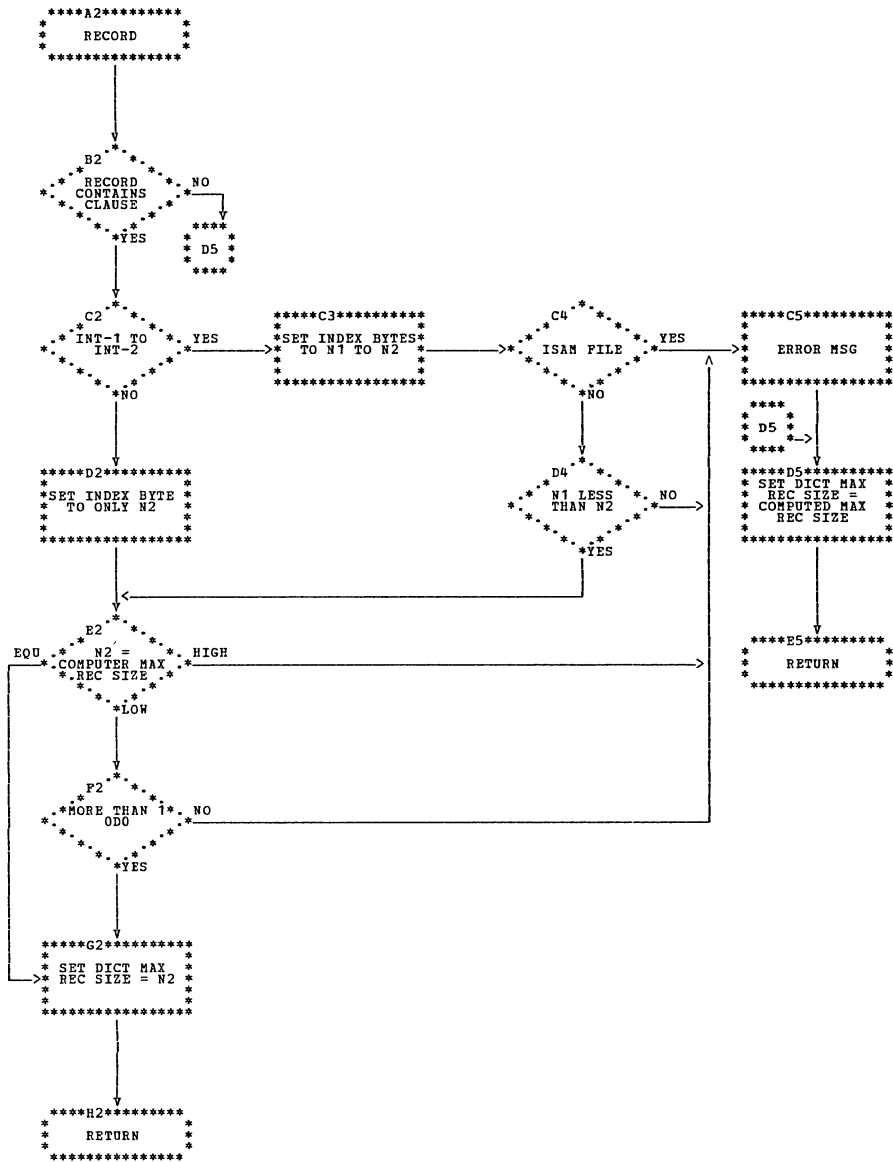


Chart GC. Phase 21: BLOCK CONTAINS Clause Processing

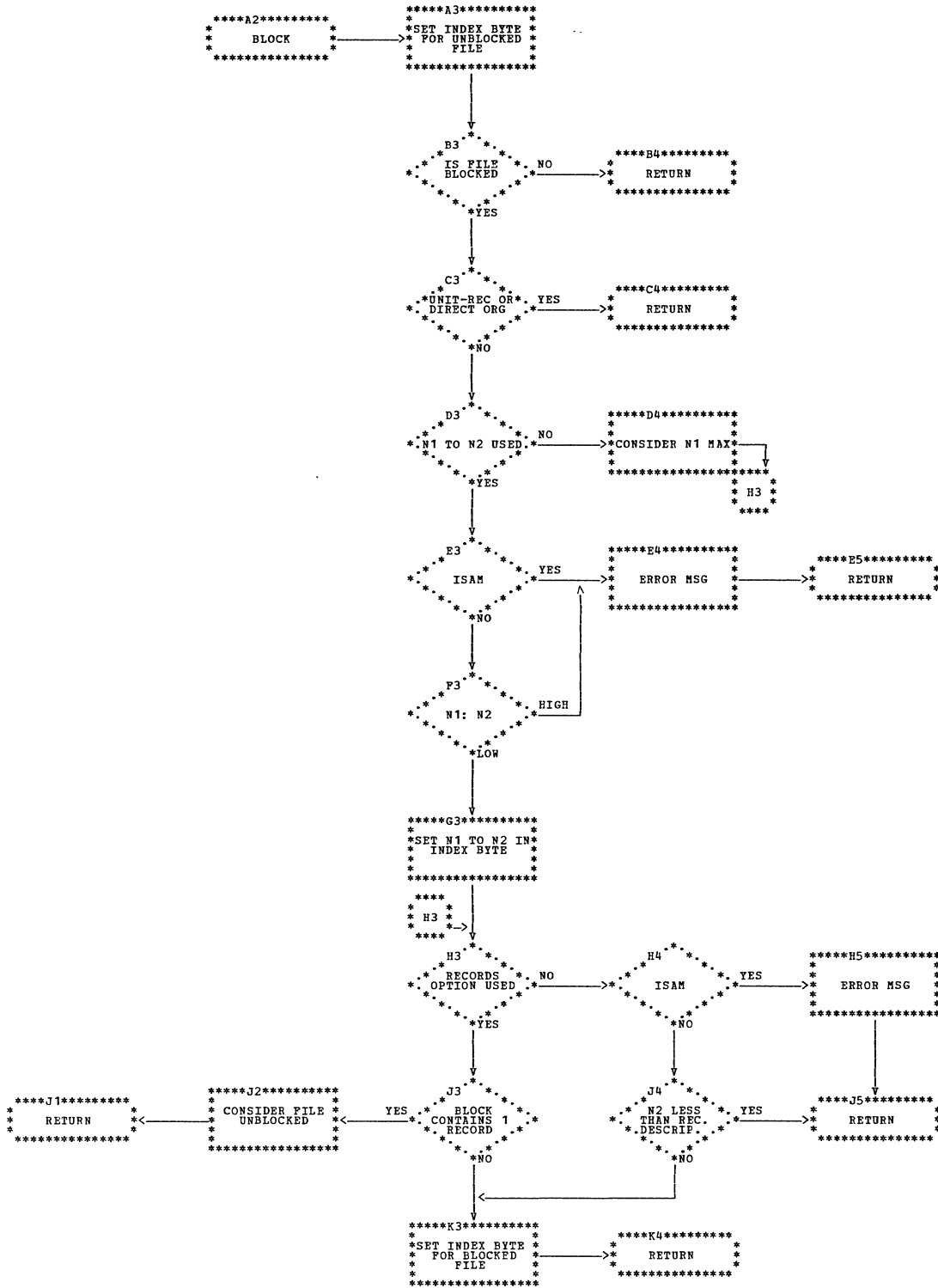


Chart GD. Phase 21: RECORDING MODE Clause Processing

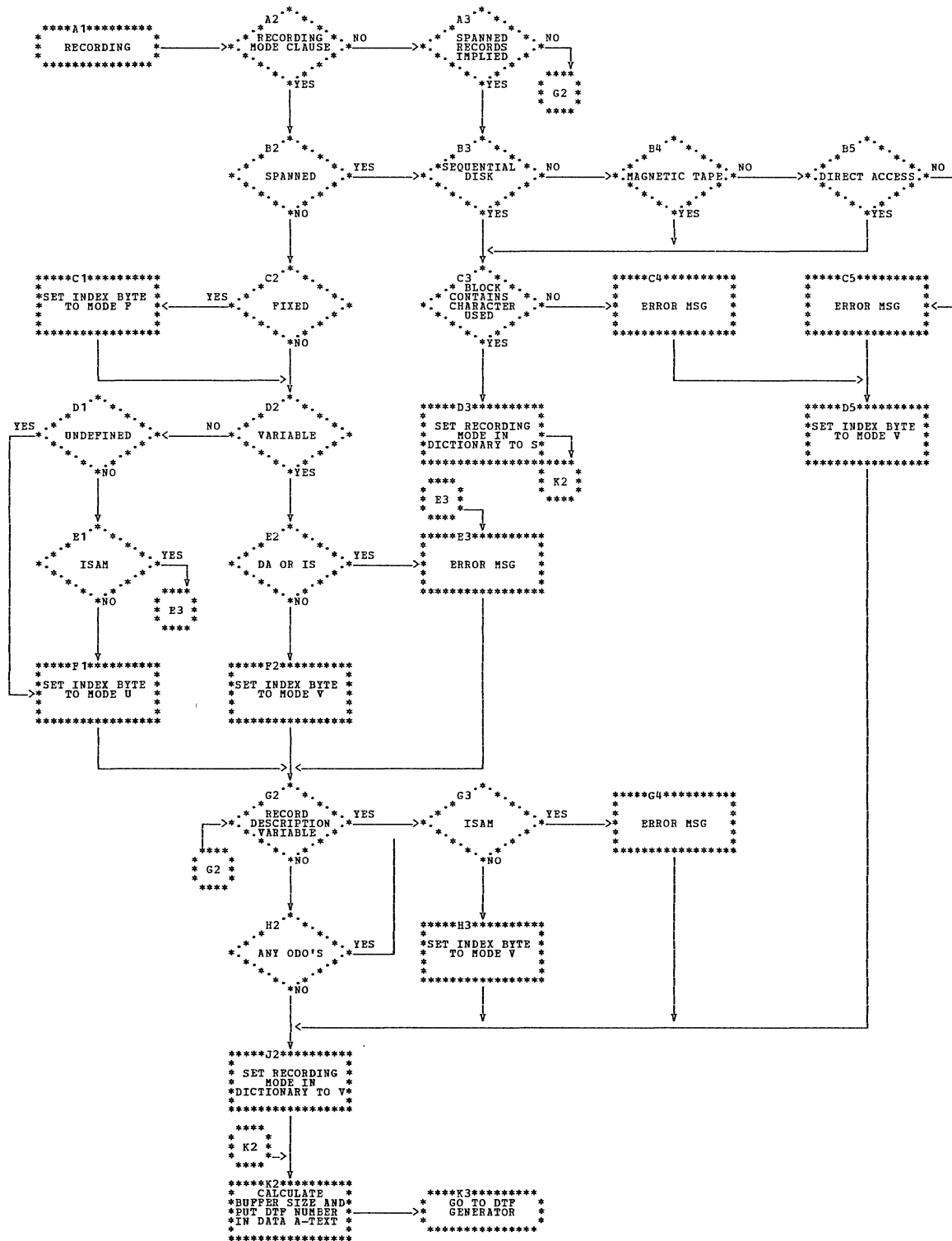


Chart GE. Phase 21: BUFGEN (Part 1 of 2)

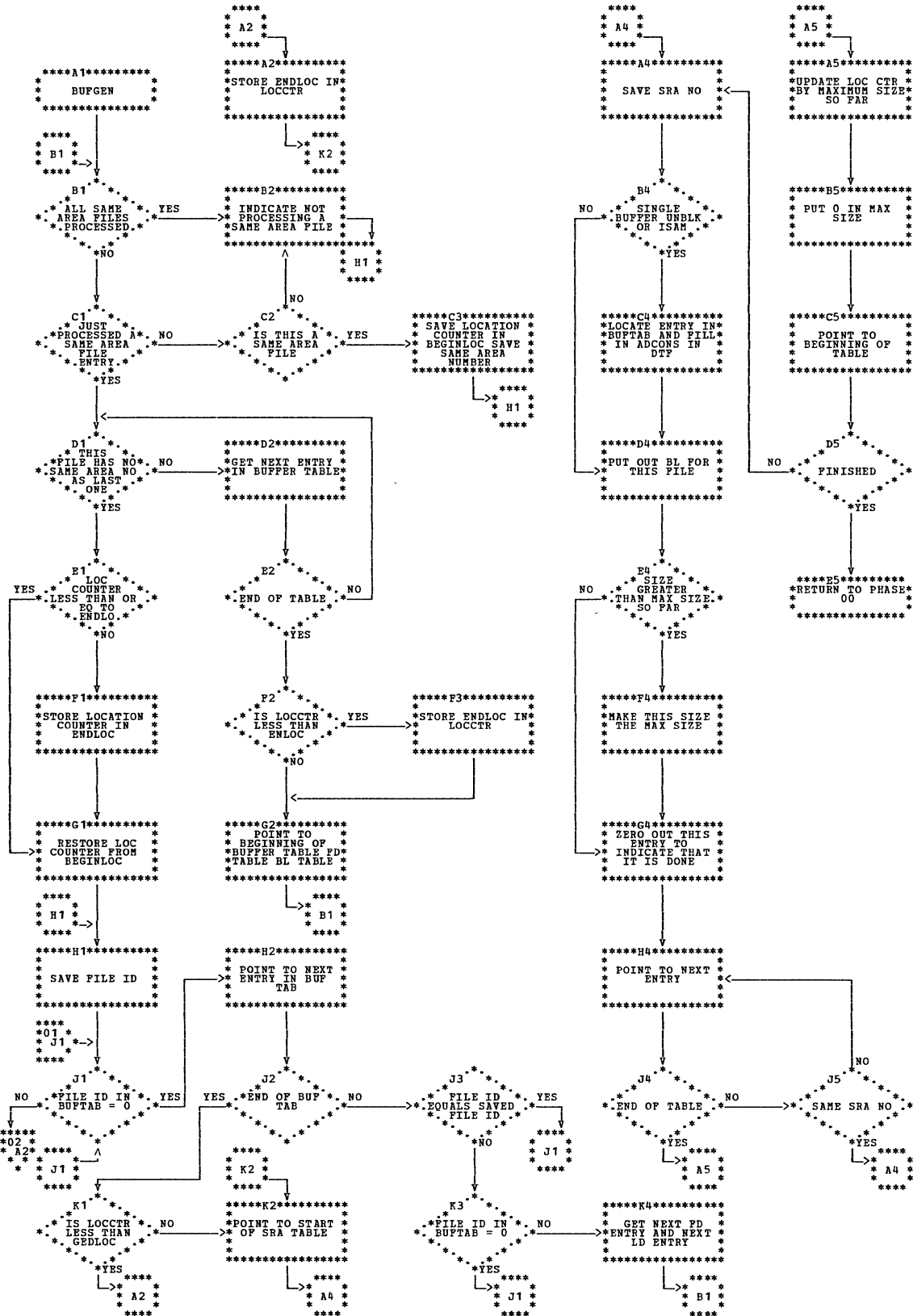




Chart GE. Phase 21: BUFGEN (Part 2 of 2)

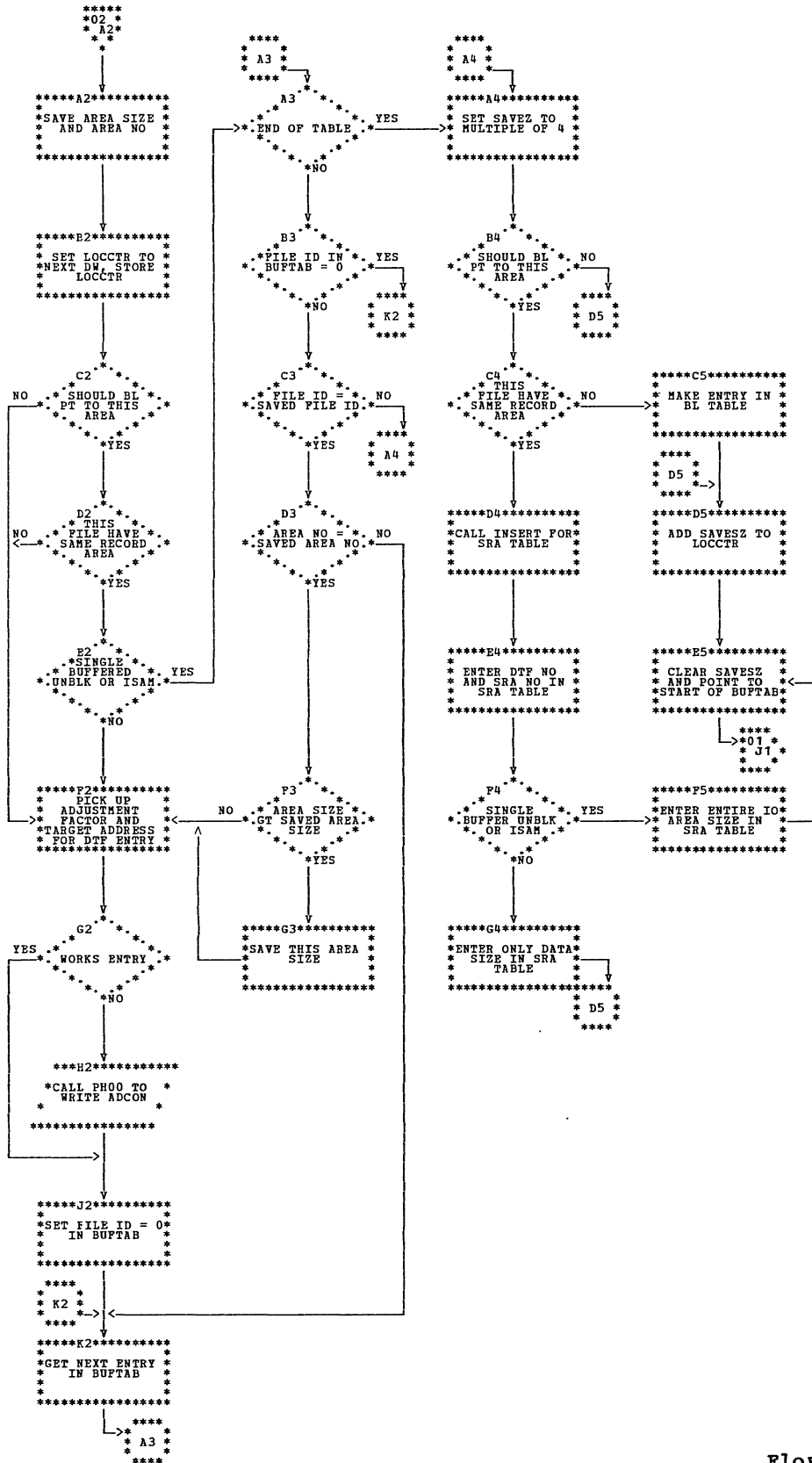


Chart HA. Phase 22 (ILACBL22): Overall Logic

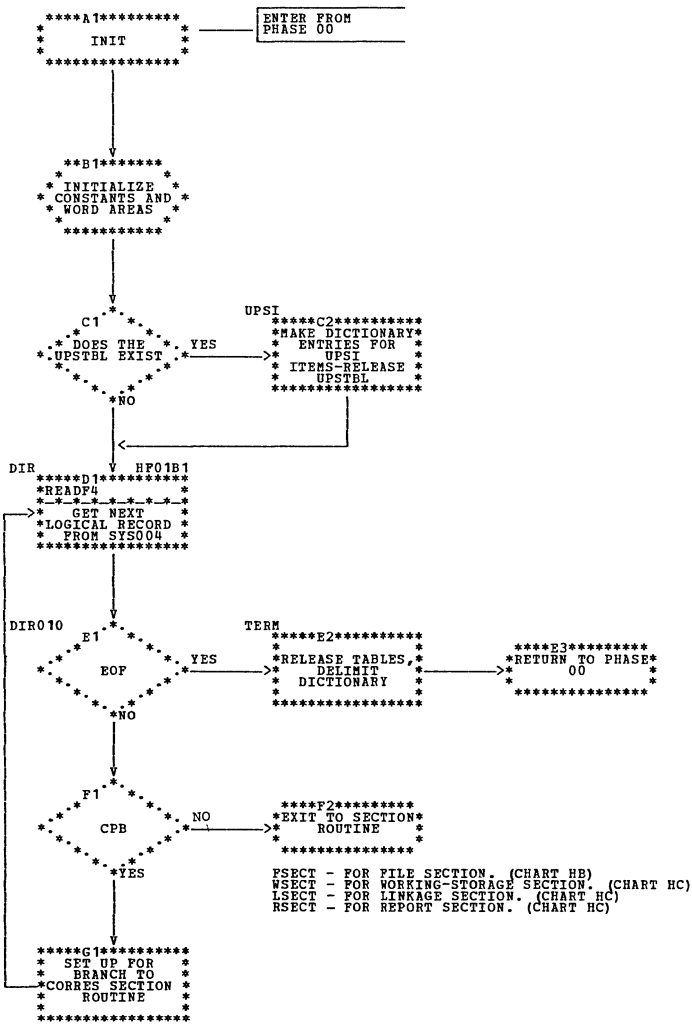


Chart HB. Phase 22: FSECT Routine

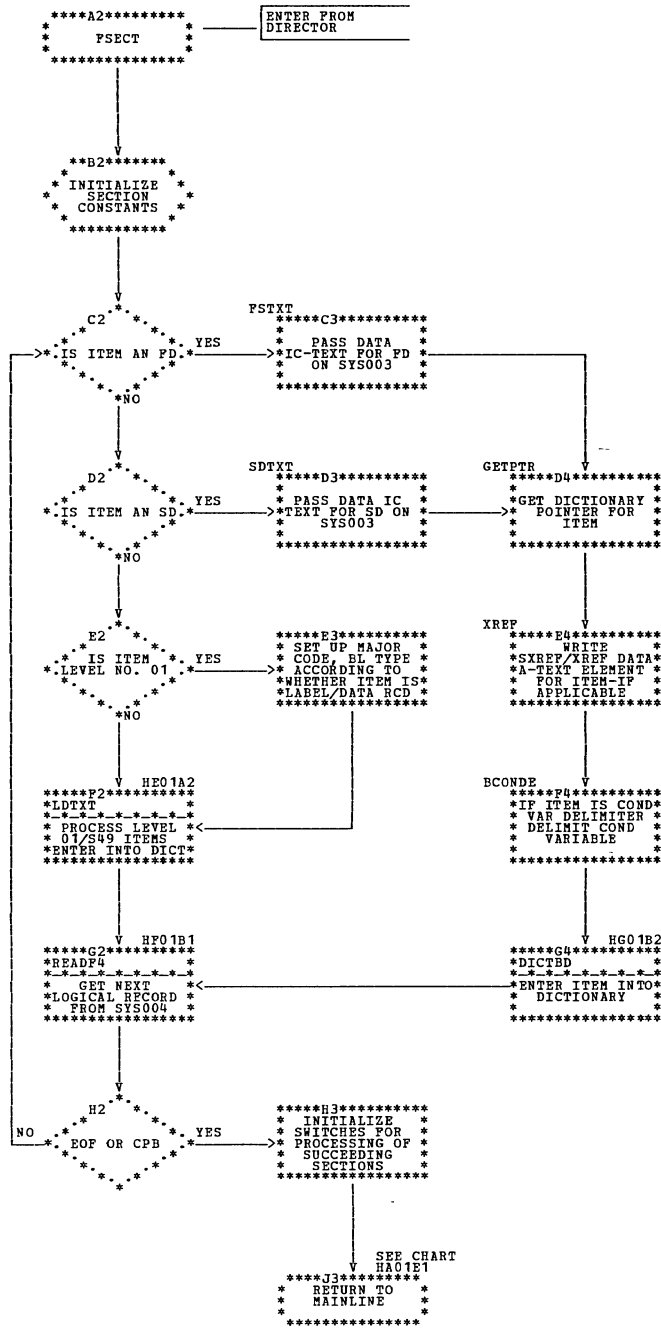


Chart HC. Phase 22: WSECT and LSECT Routines

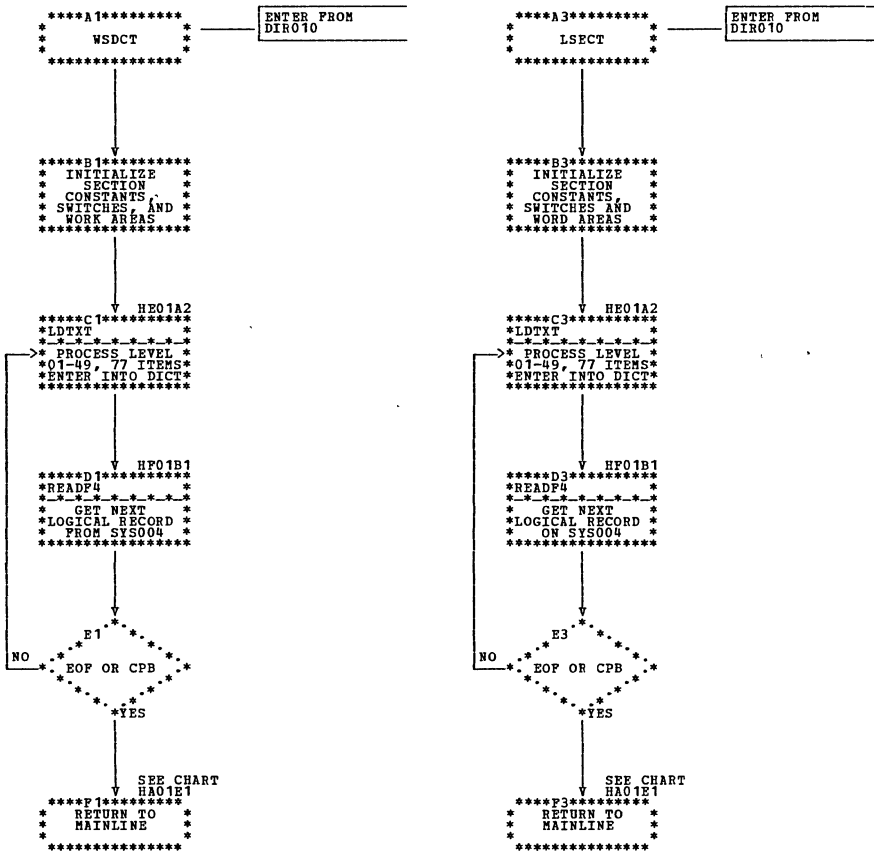


Chart HD. Phase 22: RSECT Routine

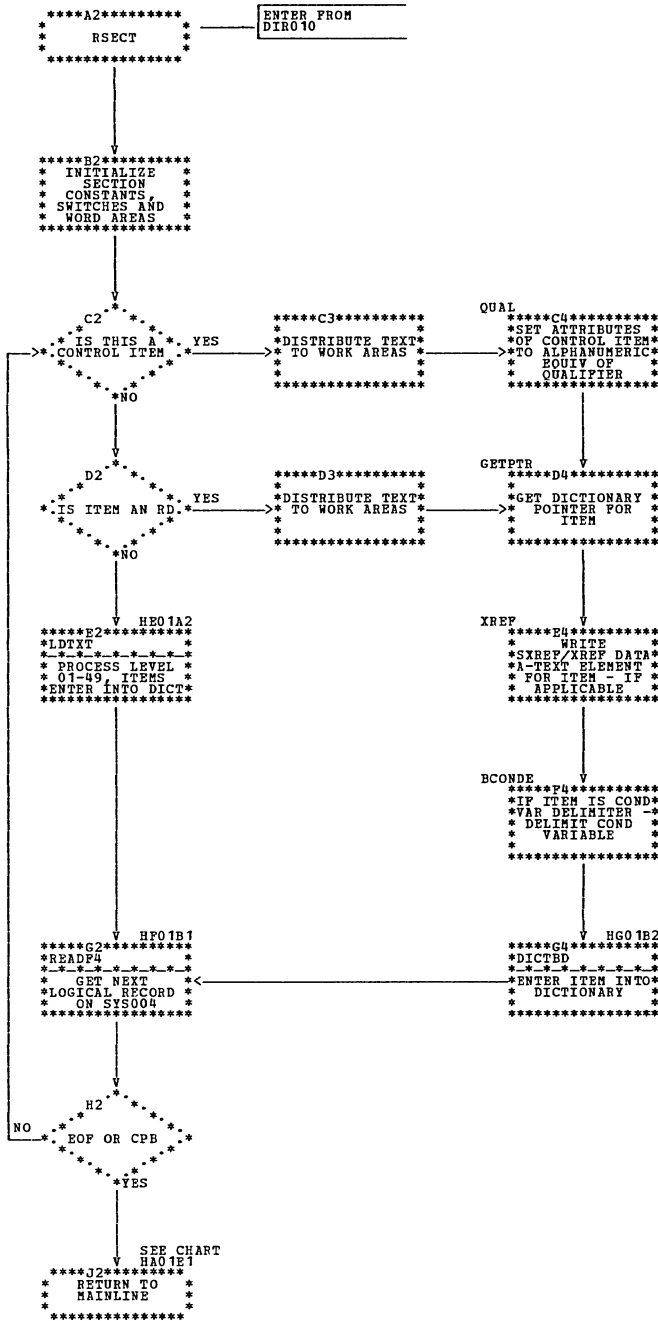


Chart HE. Phase 22: LDTXT Routine

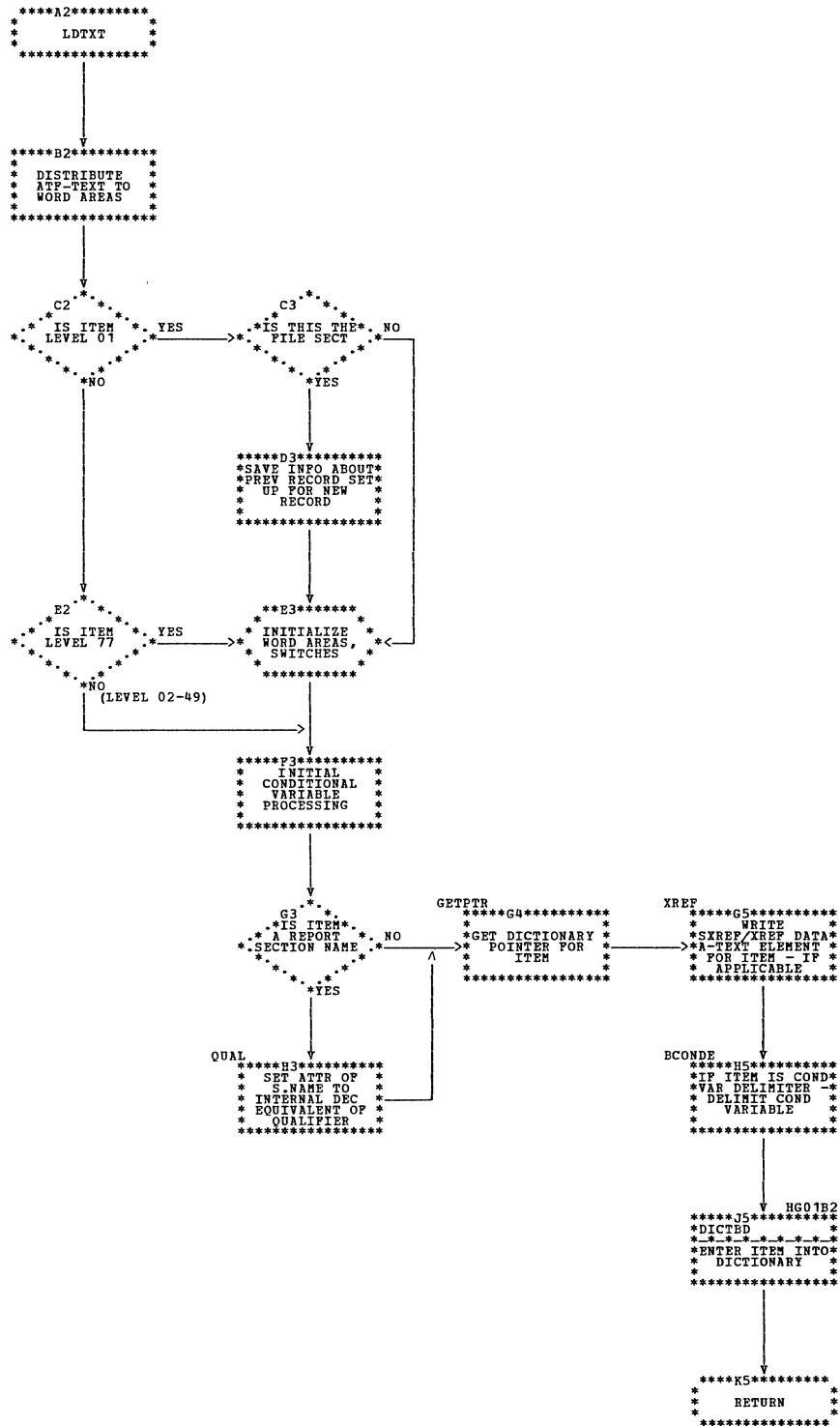


Chart HF. Phase 22: READF4 Routine

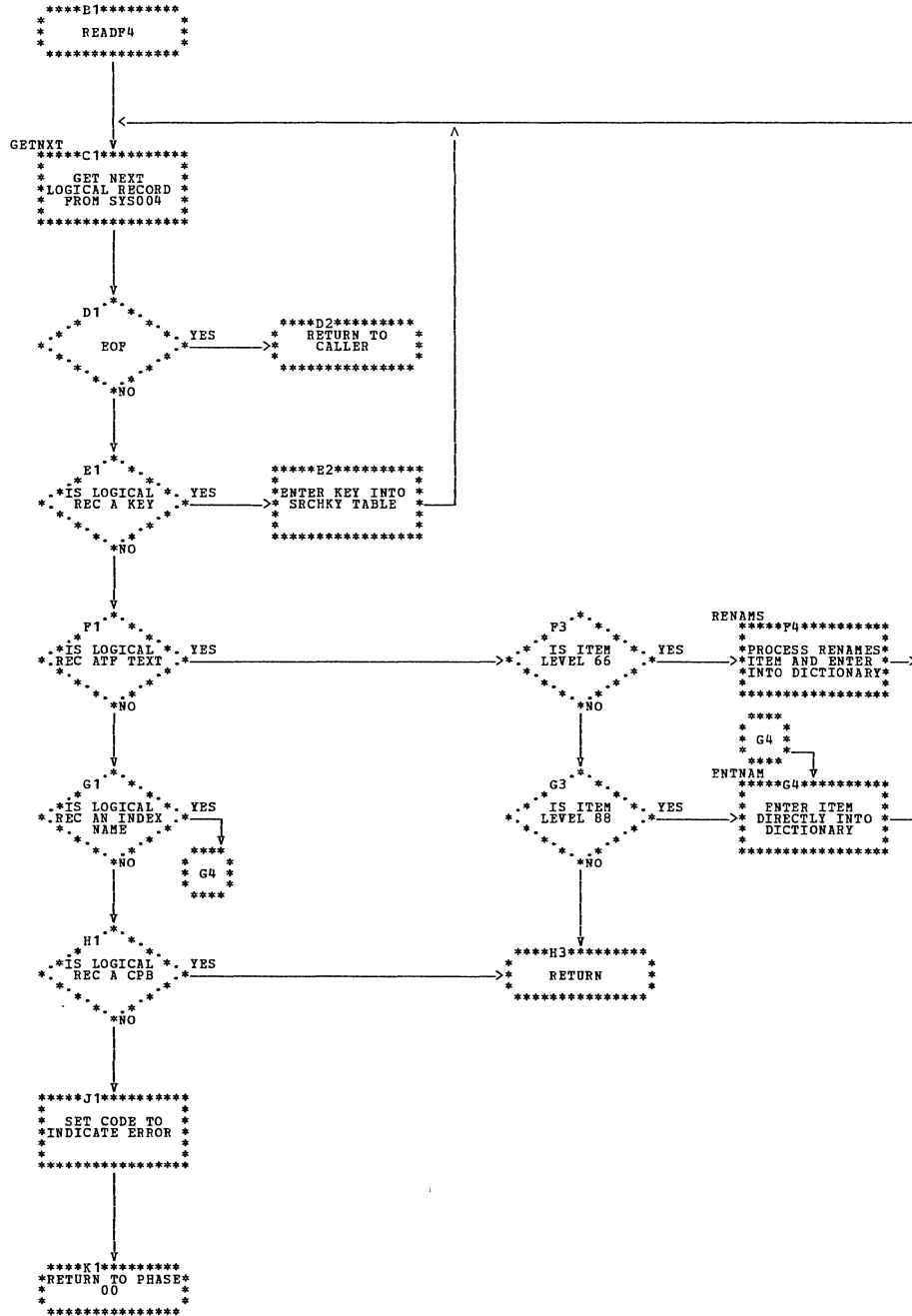


Chart HG. Phase 22: DICTBD Routine

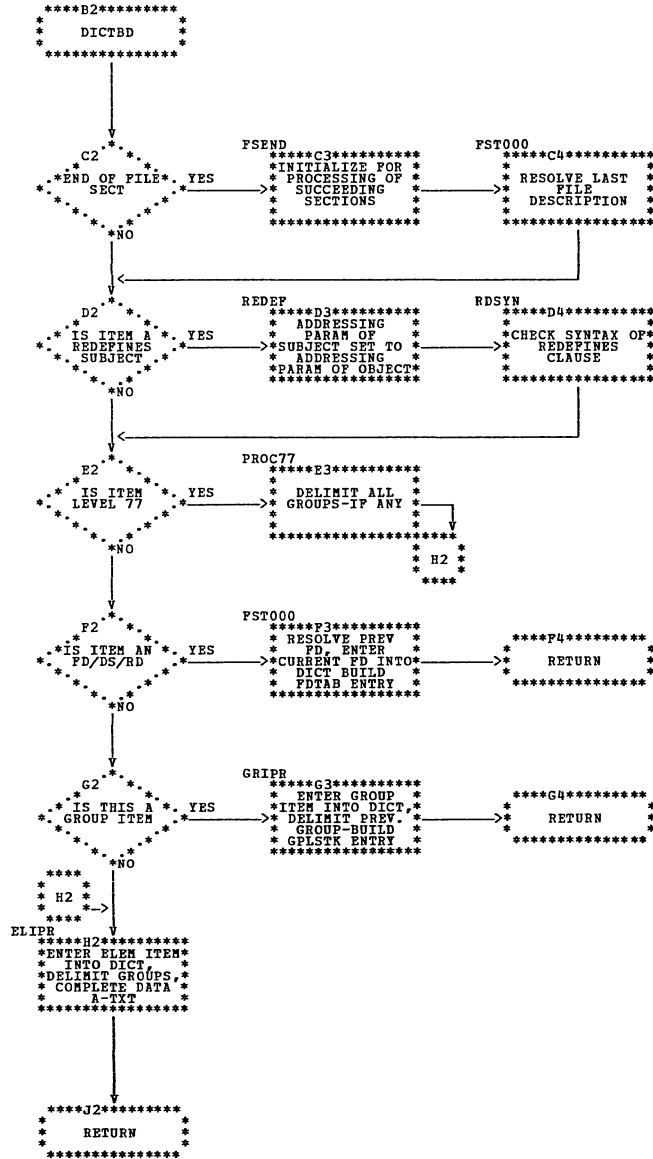




Chart IA. Phase 25: Overall Logic

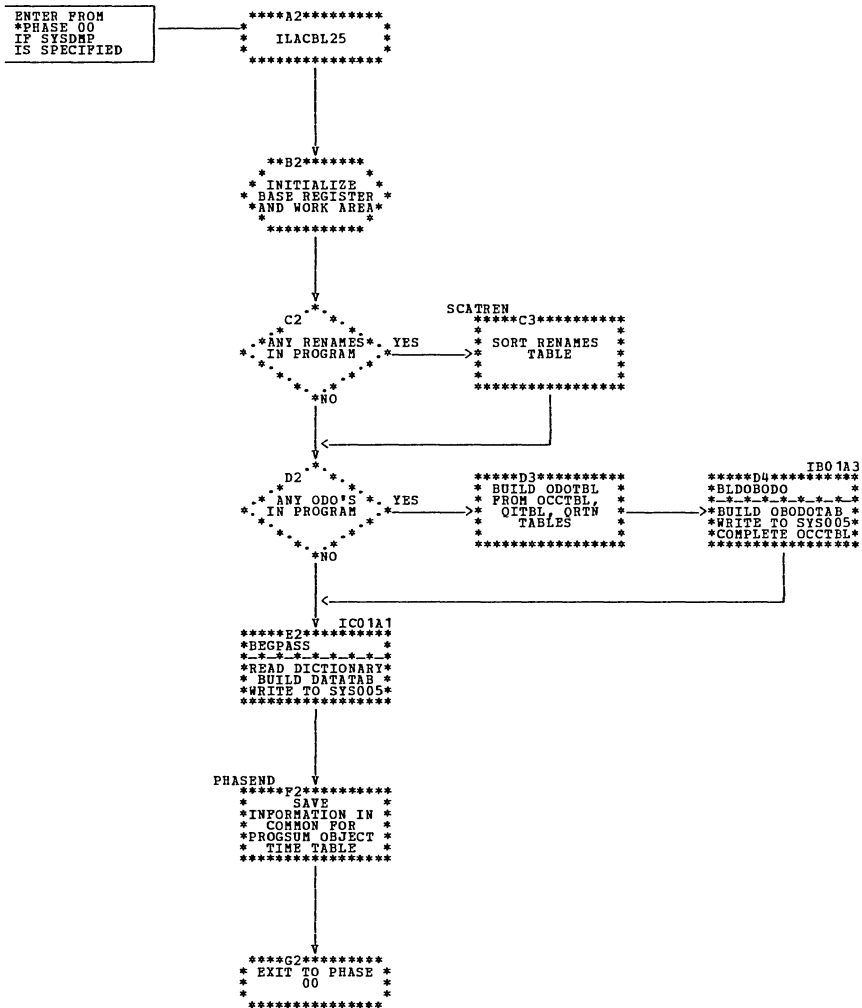


Chart IB. Phase 25: ODOBLD, BLD0BODO, and ENPP1

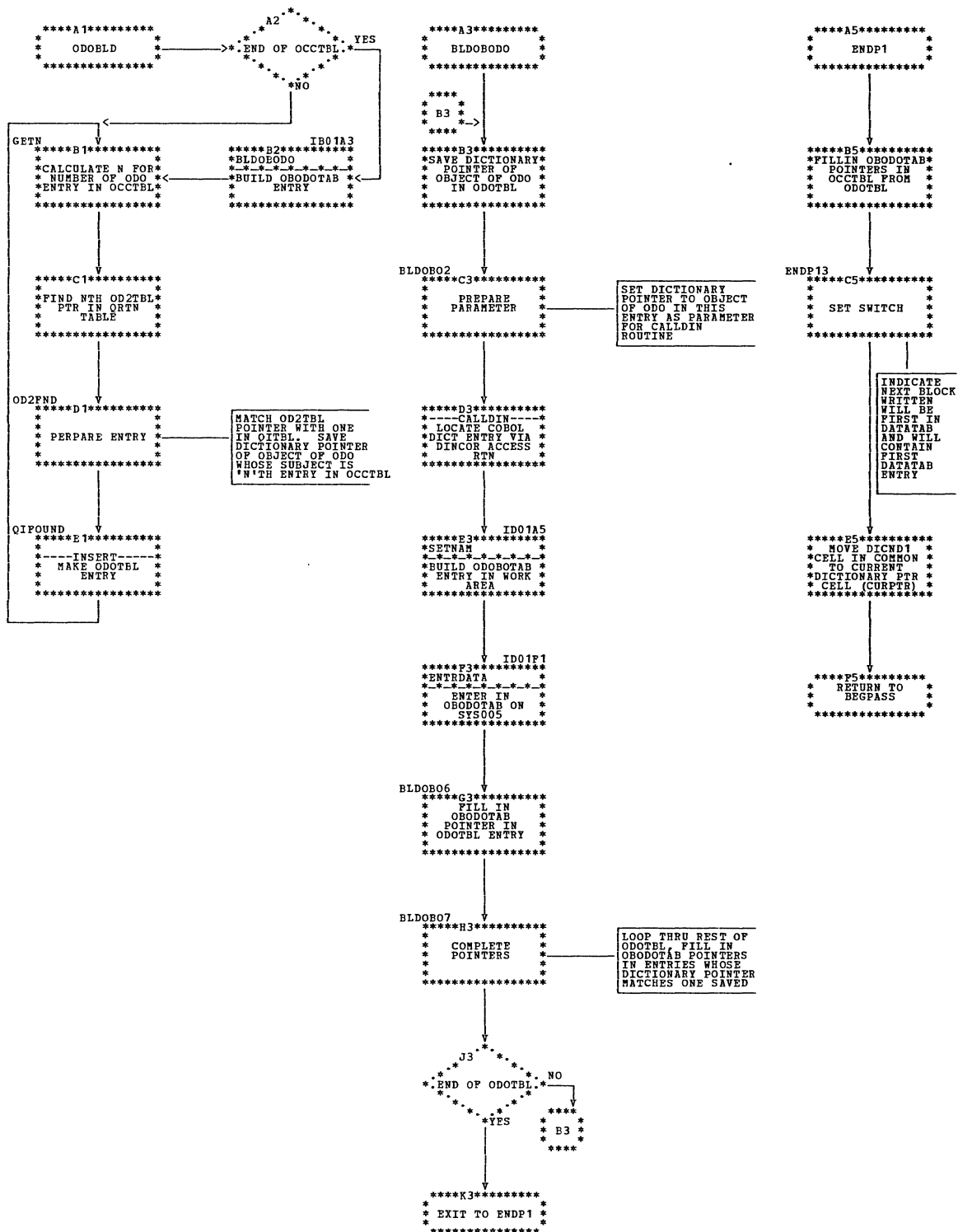


Chart IC. Phase 25: BEGPASS

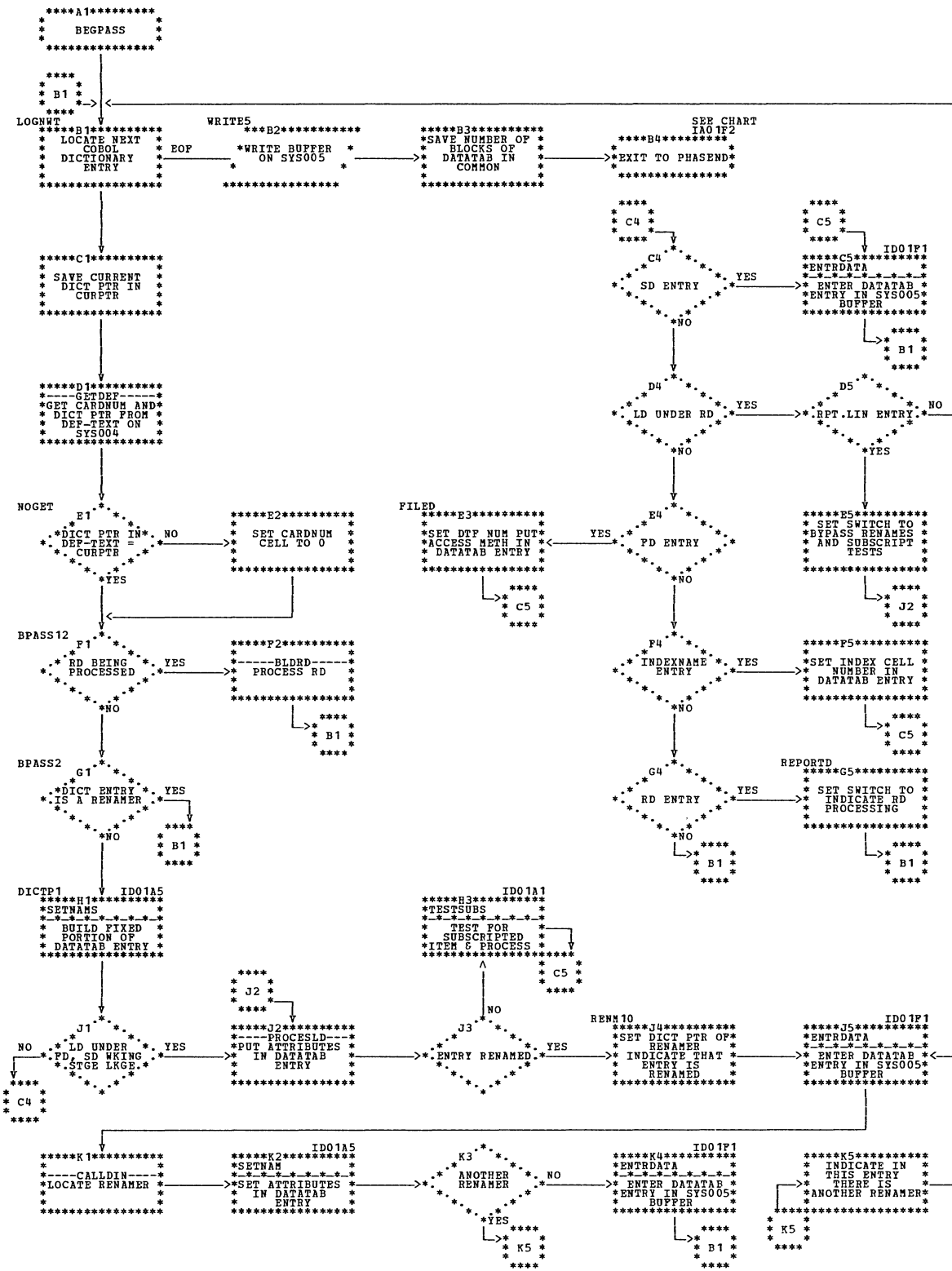


Chart ID. Phase 25: TESTSUBS and SETNAMS

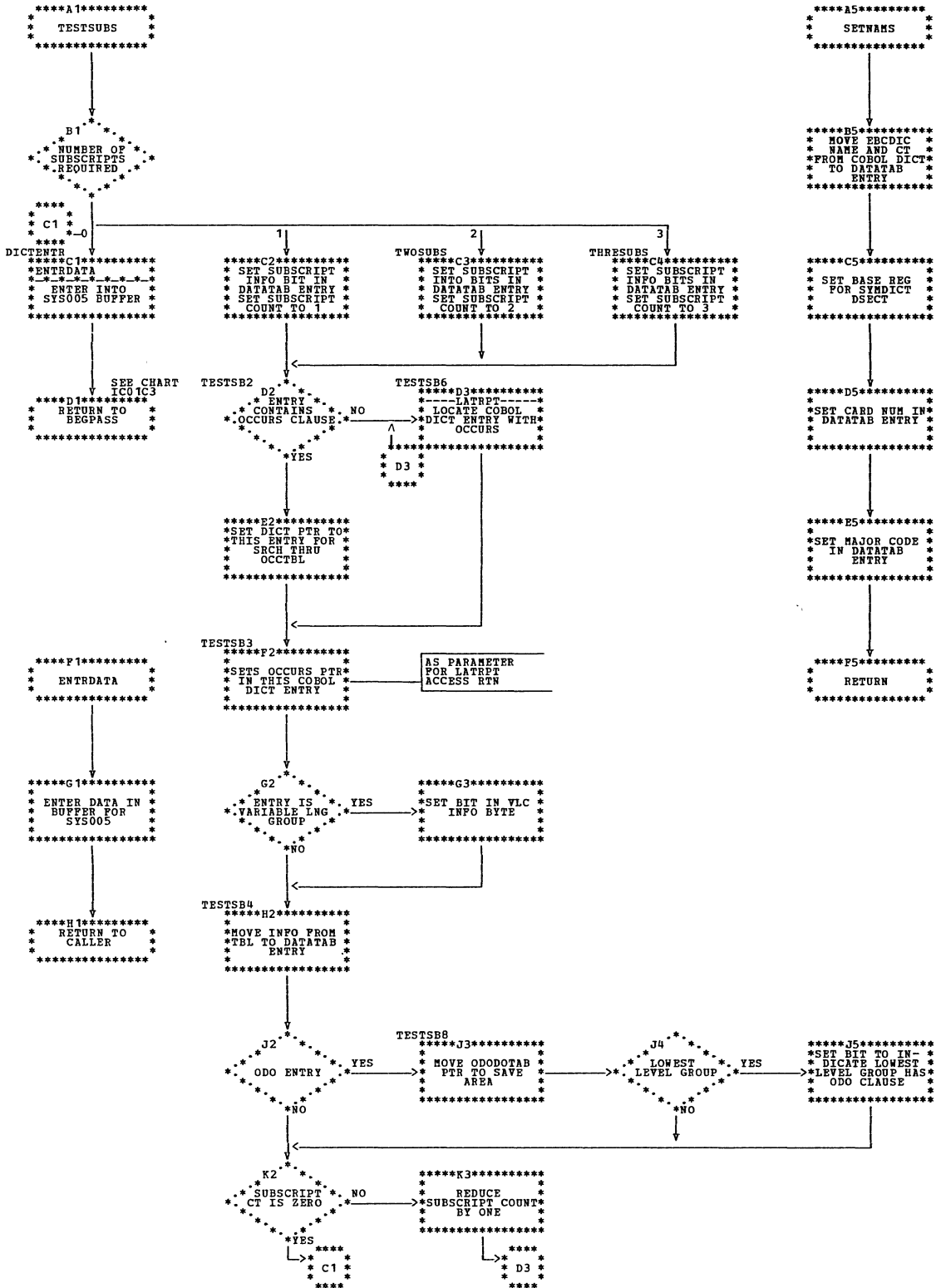


Chart JA. Phase 30 (ILACBL30): Overall Logic

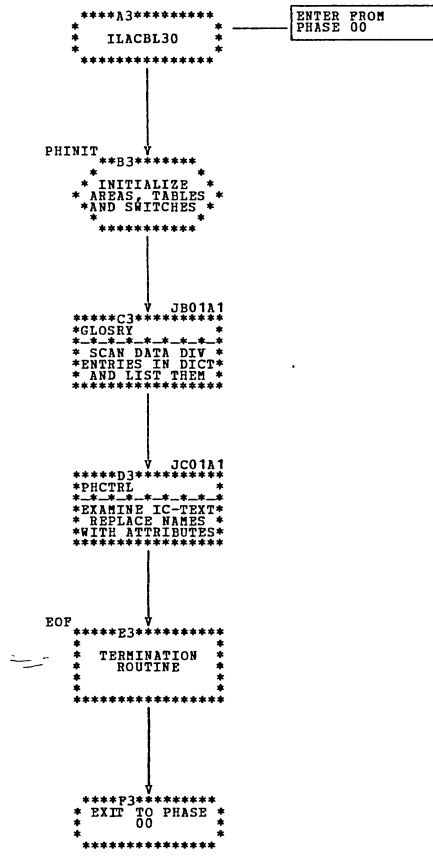


Chart JB. Phase 30: GLOSRY Routine

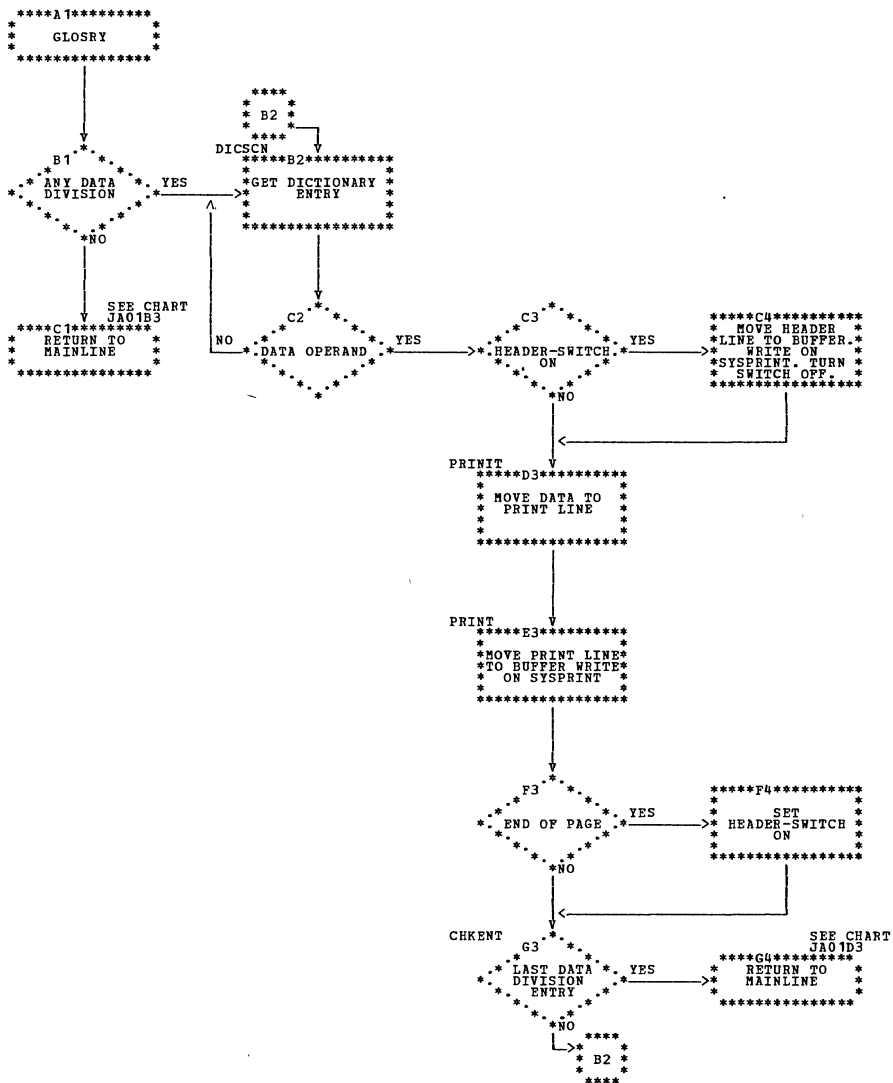


Chart JC. Phase 30: PHCTRL Routine

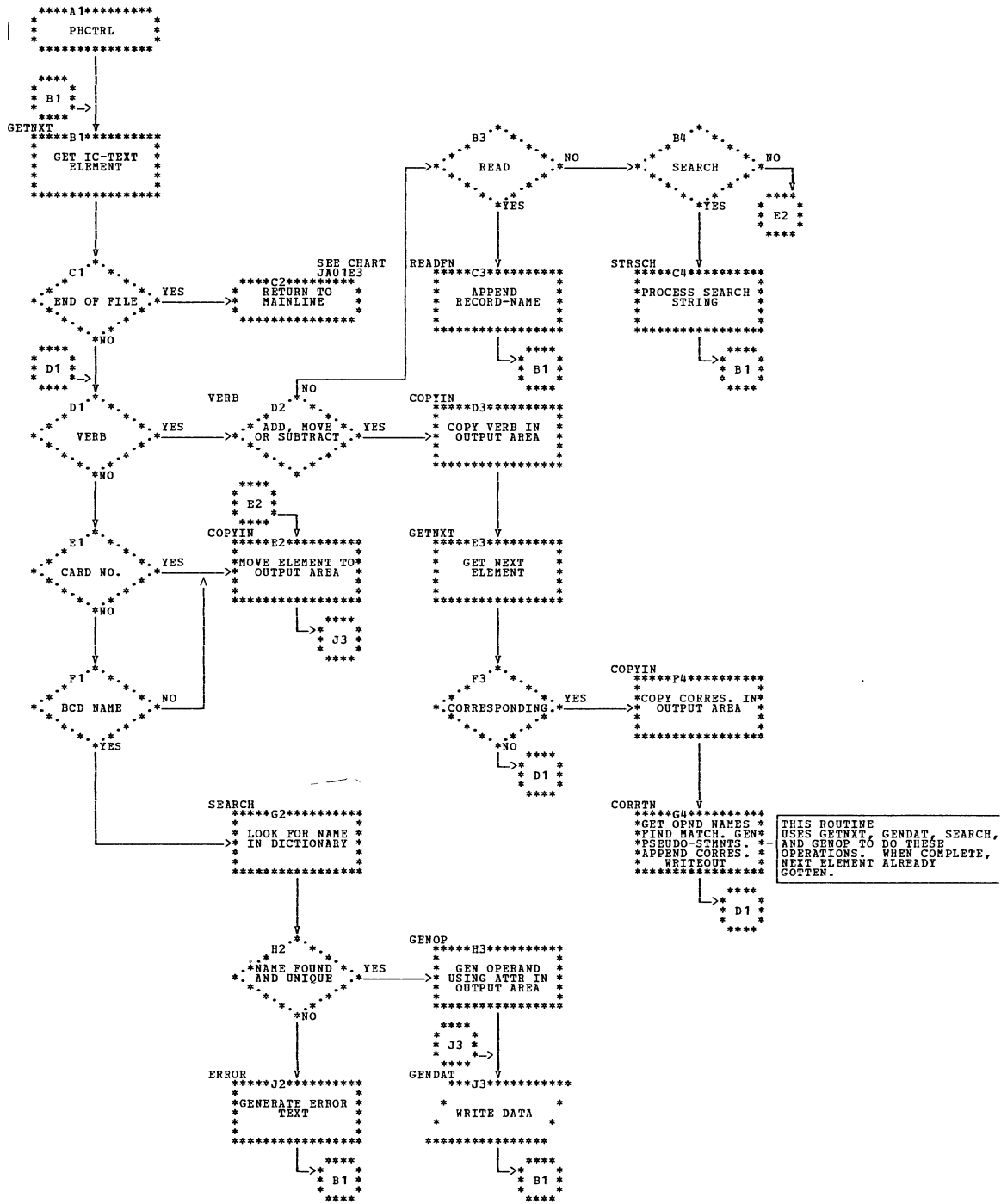


Chart KA. Phase 40 (ILACBL40): Overall Logic

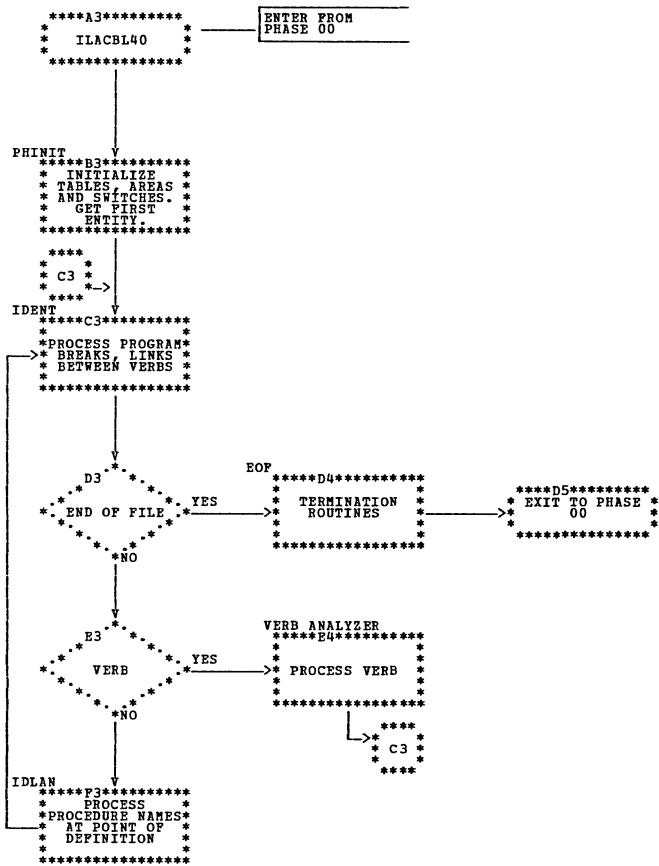




Chart KB. Phase 40: IF Processing

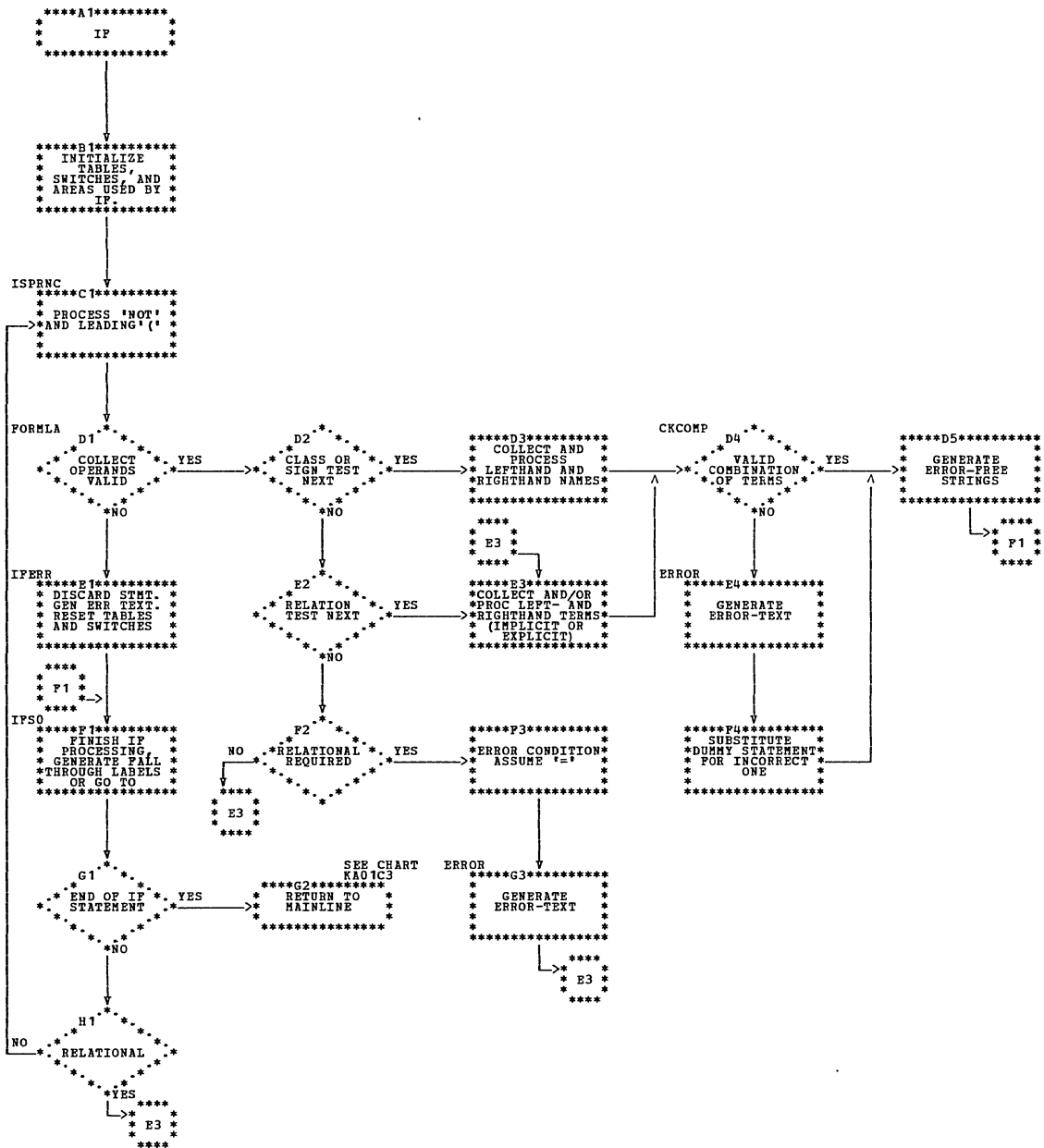


Chart KC. Phase 40: PERFORM Processing

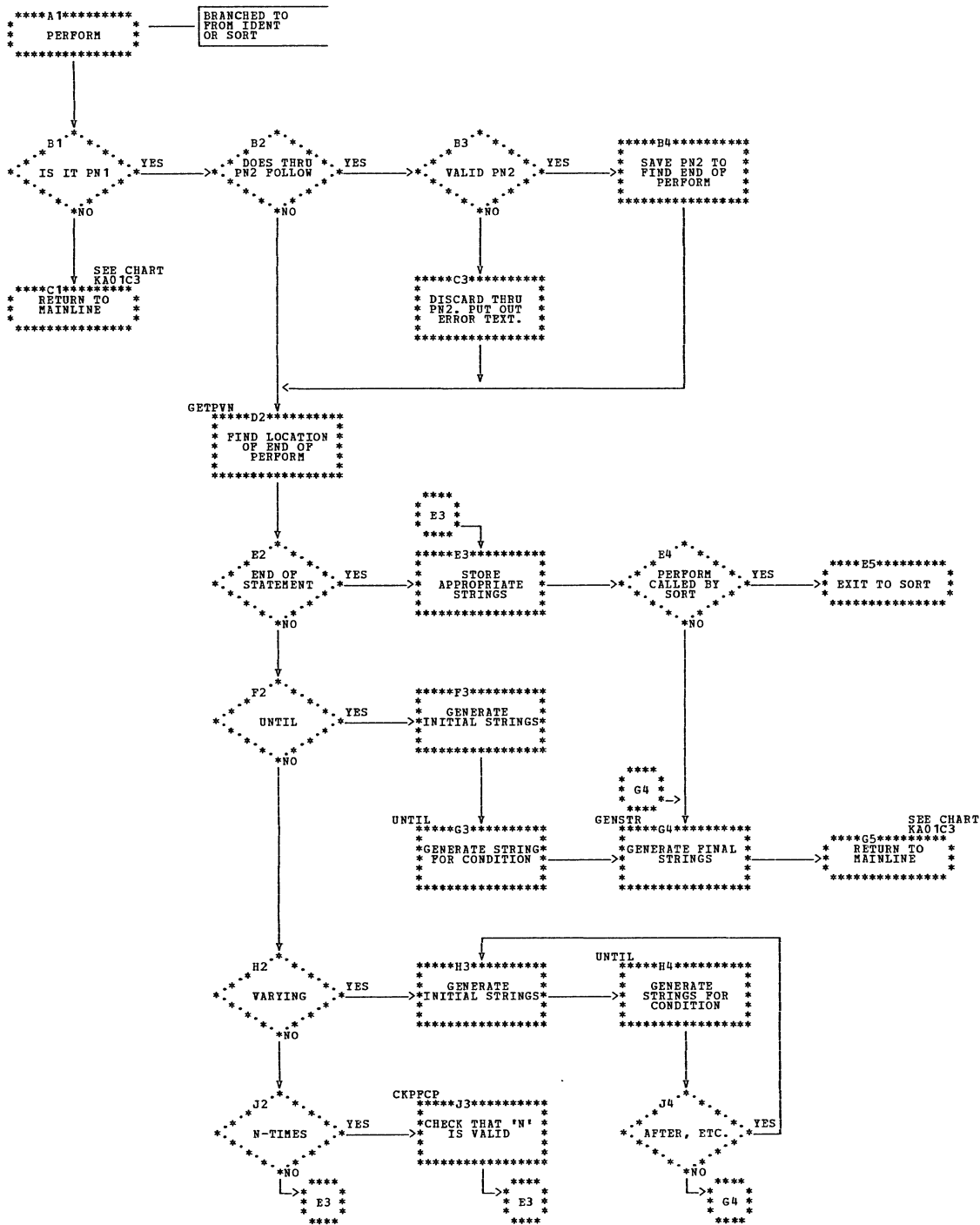


Chart LA. Phase 50 (ILACBL50): Overall Logic

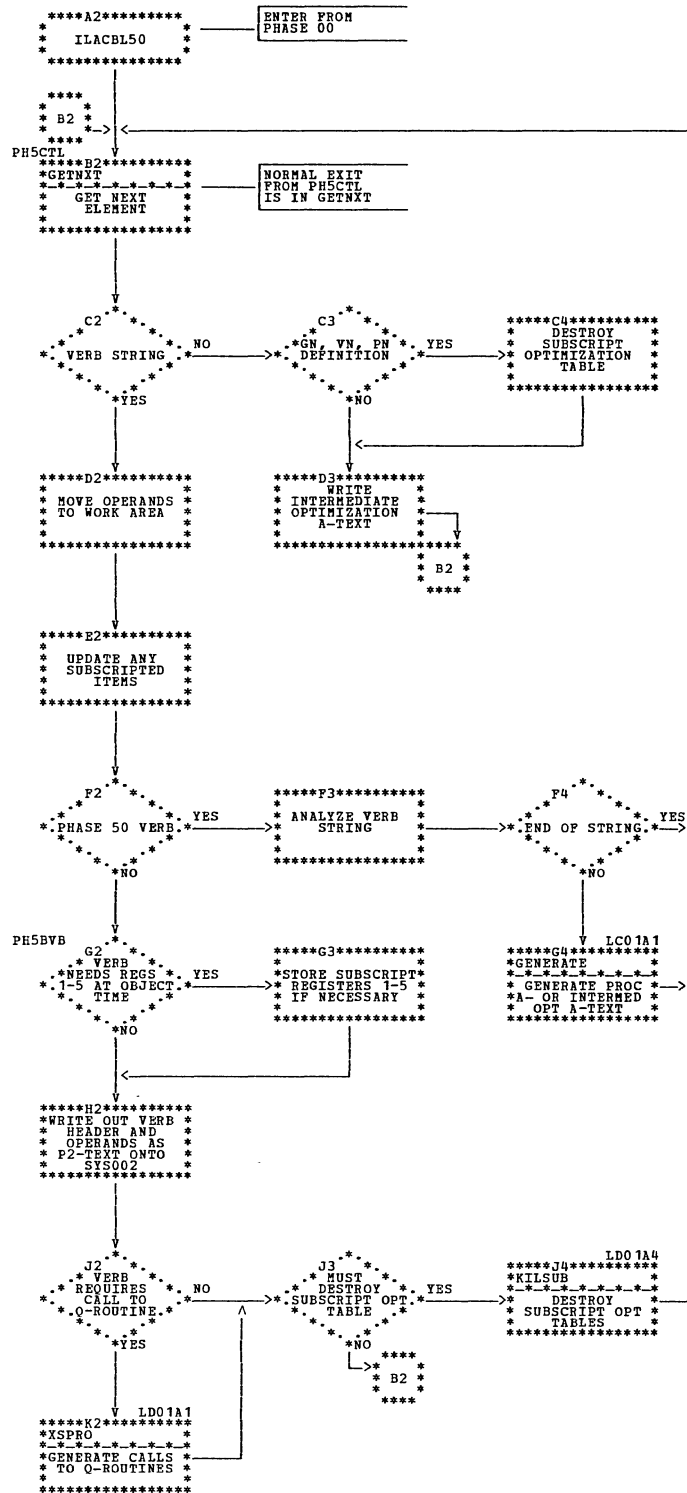


Chart LB. Phase 50: GETNXT (Part 1 of 2)

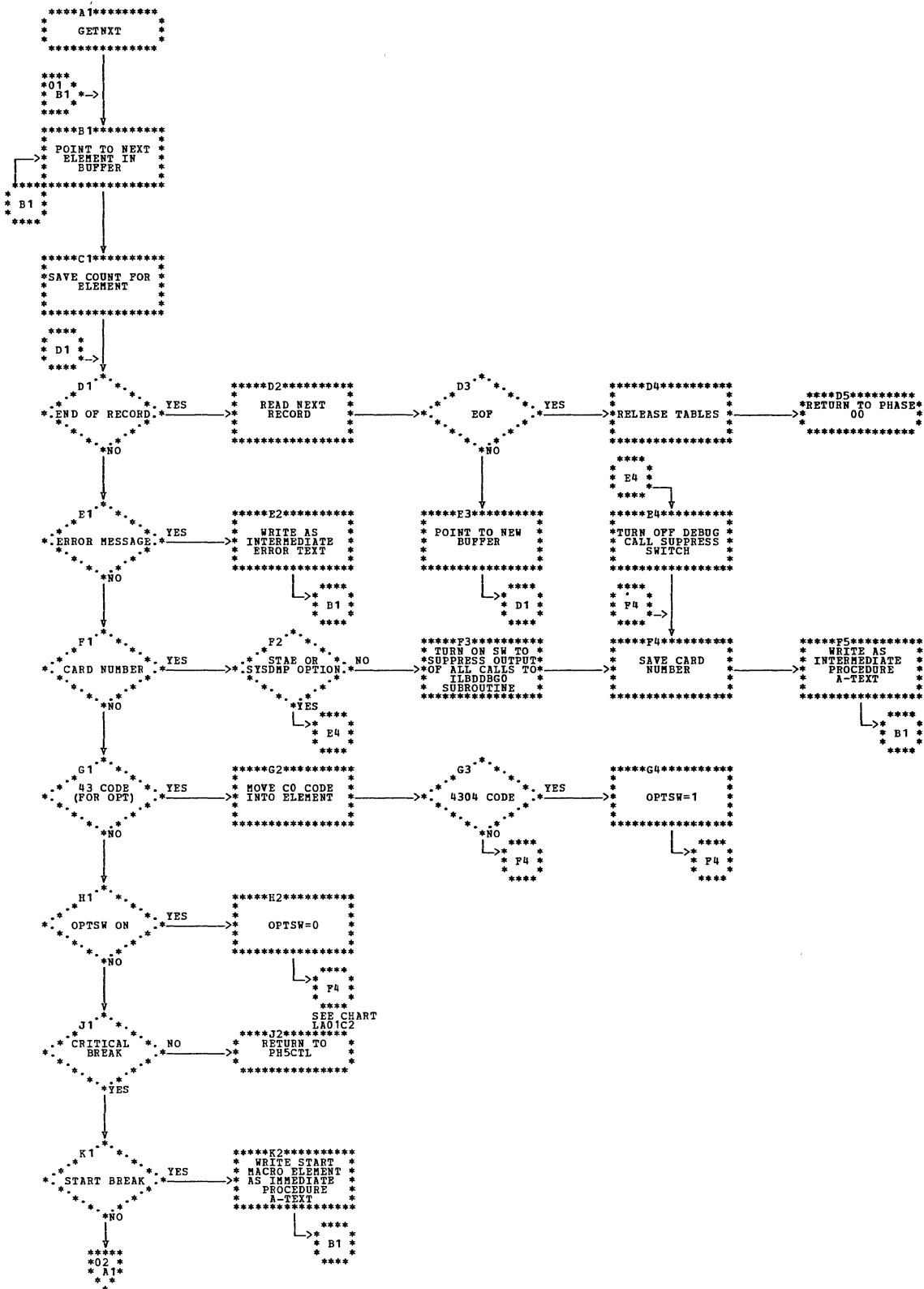


Chart LB. Phase 50: GETNXT (Part 2 of 2)

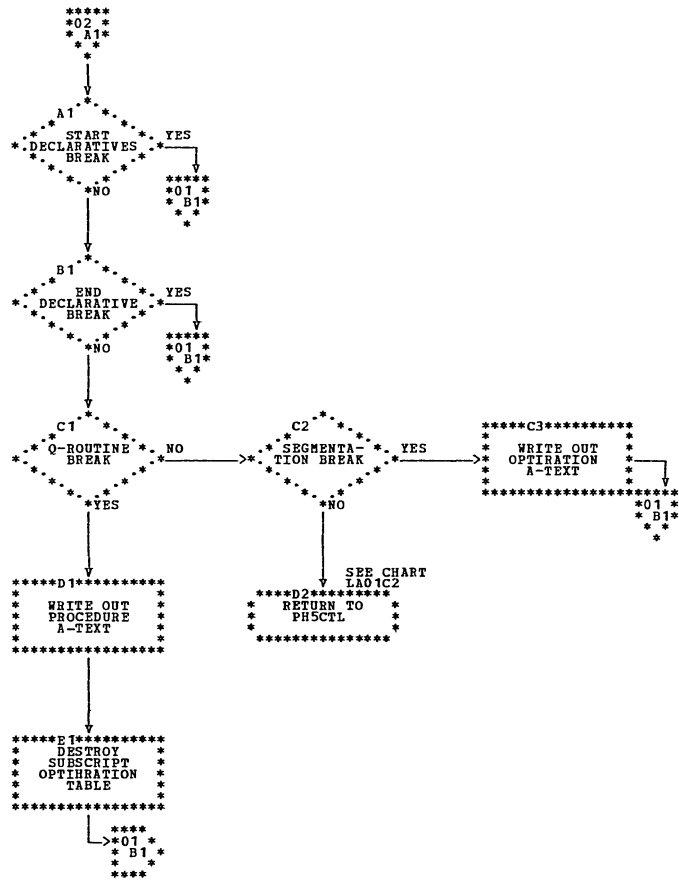


Chart LC. Phase 50: A-text Generator

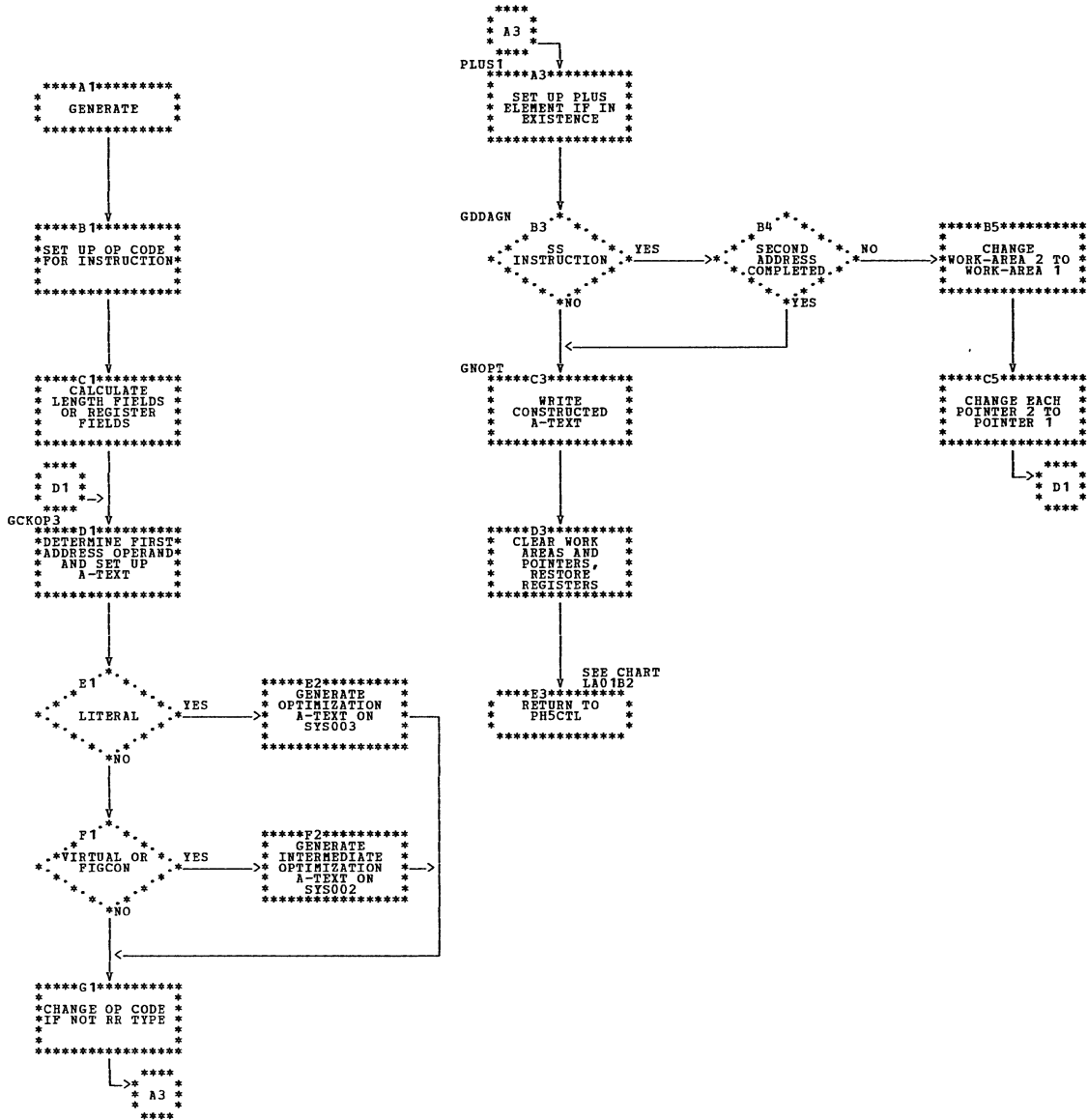


Chart LD. Phase 50: XSPRO and KILSUB Routines

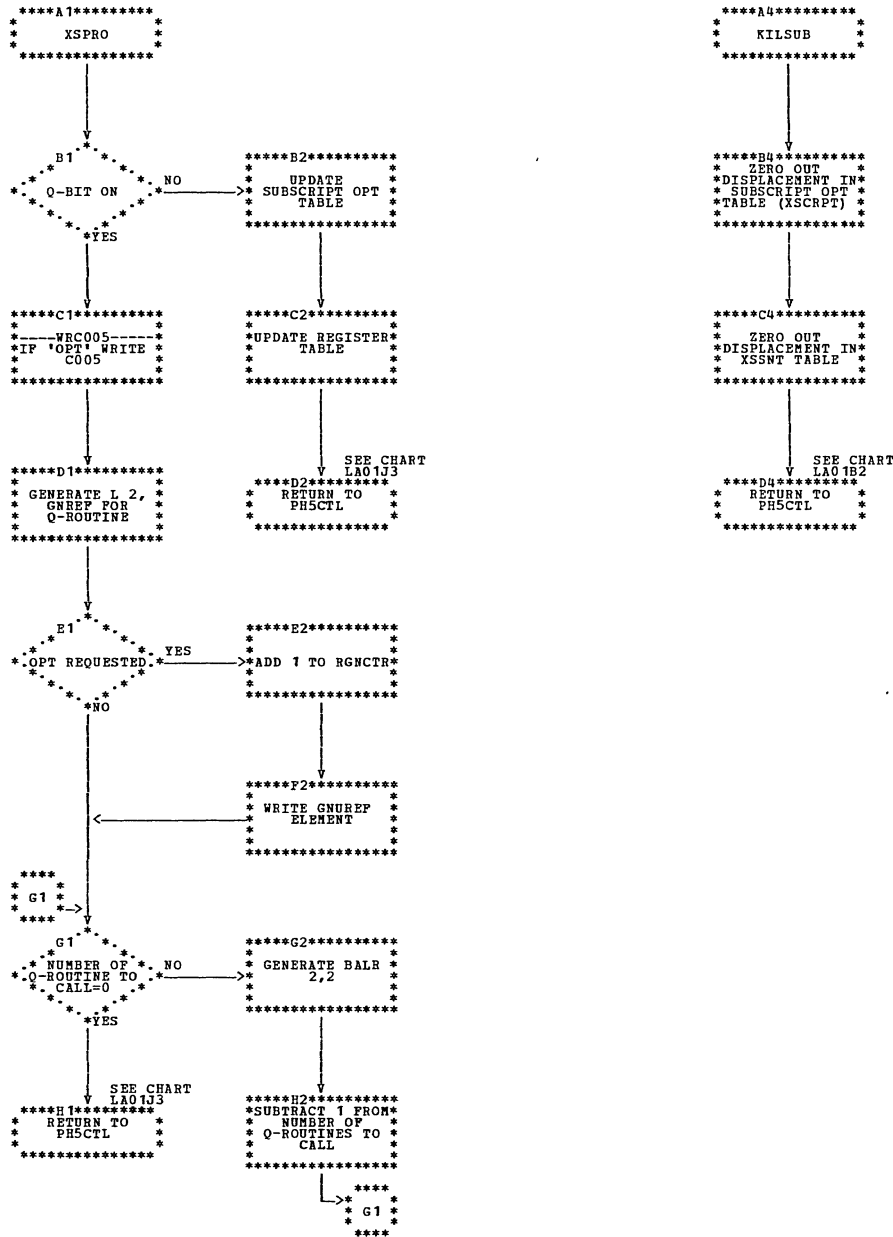


Chart LE. Phase 50: DRGTEST

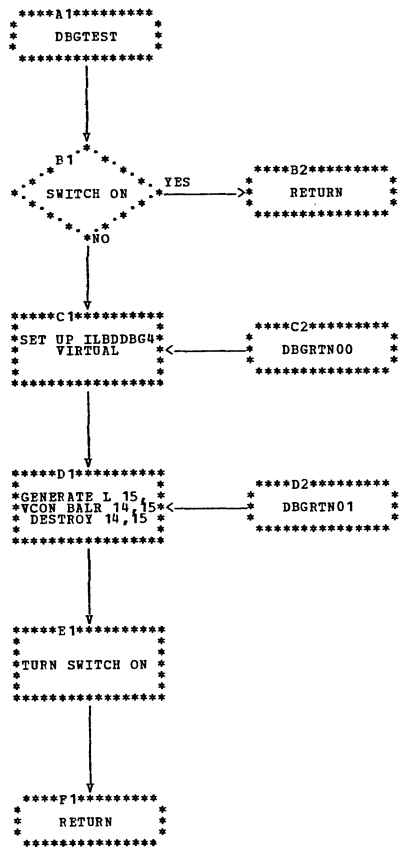




Chart MA. Phase 51 (ILACBL51): Overall Logic

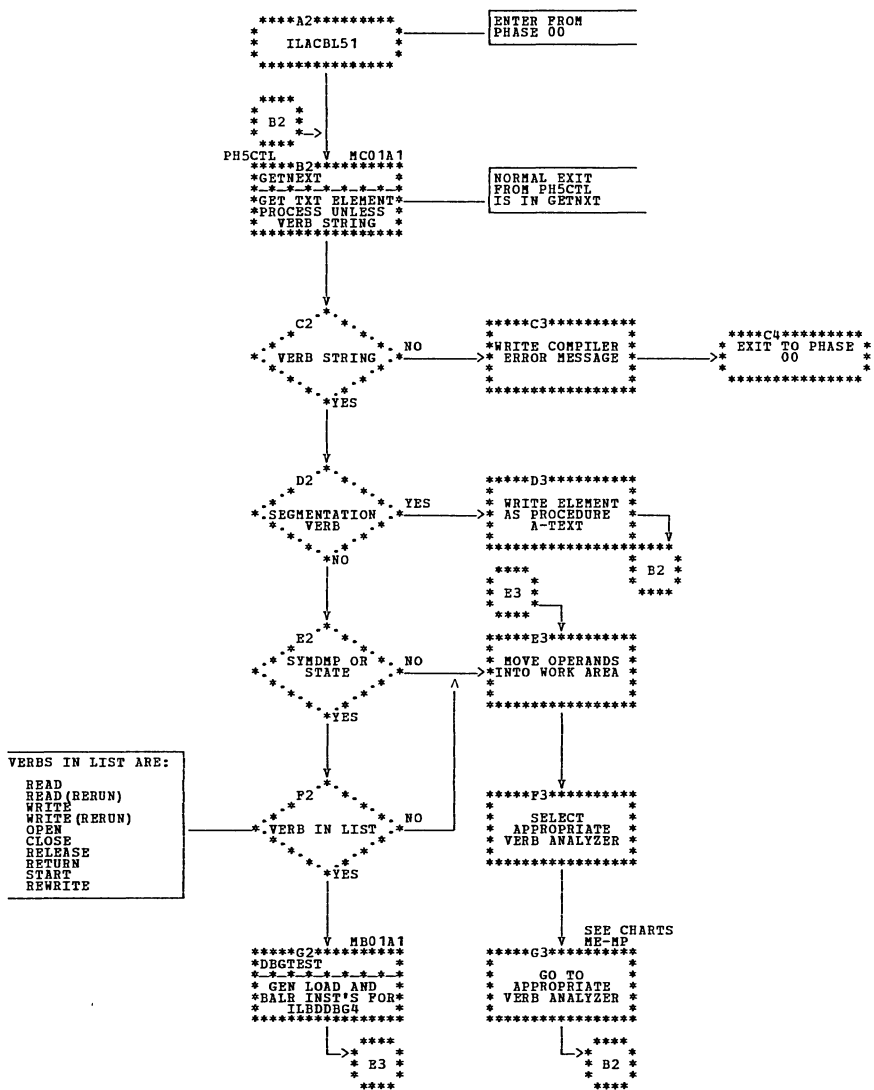


Chart MB. Phase 51: DBGTEST

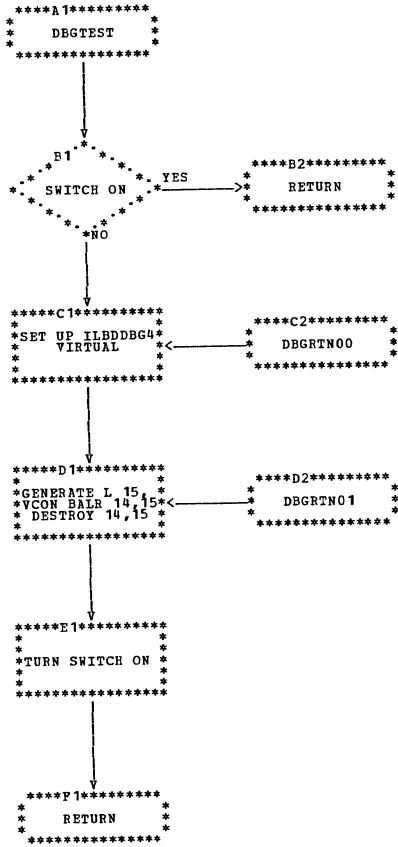


Chart MC. Phase 51: GETNXT Routine

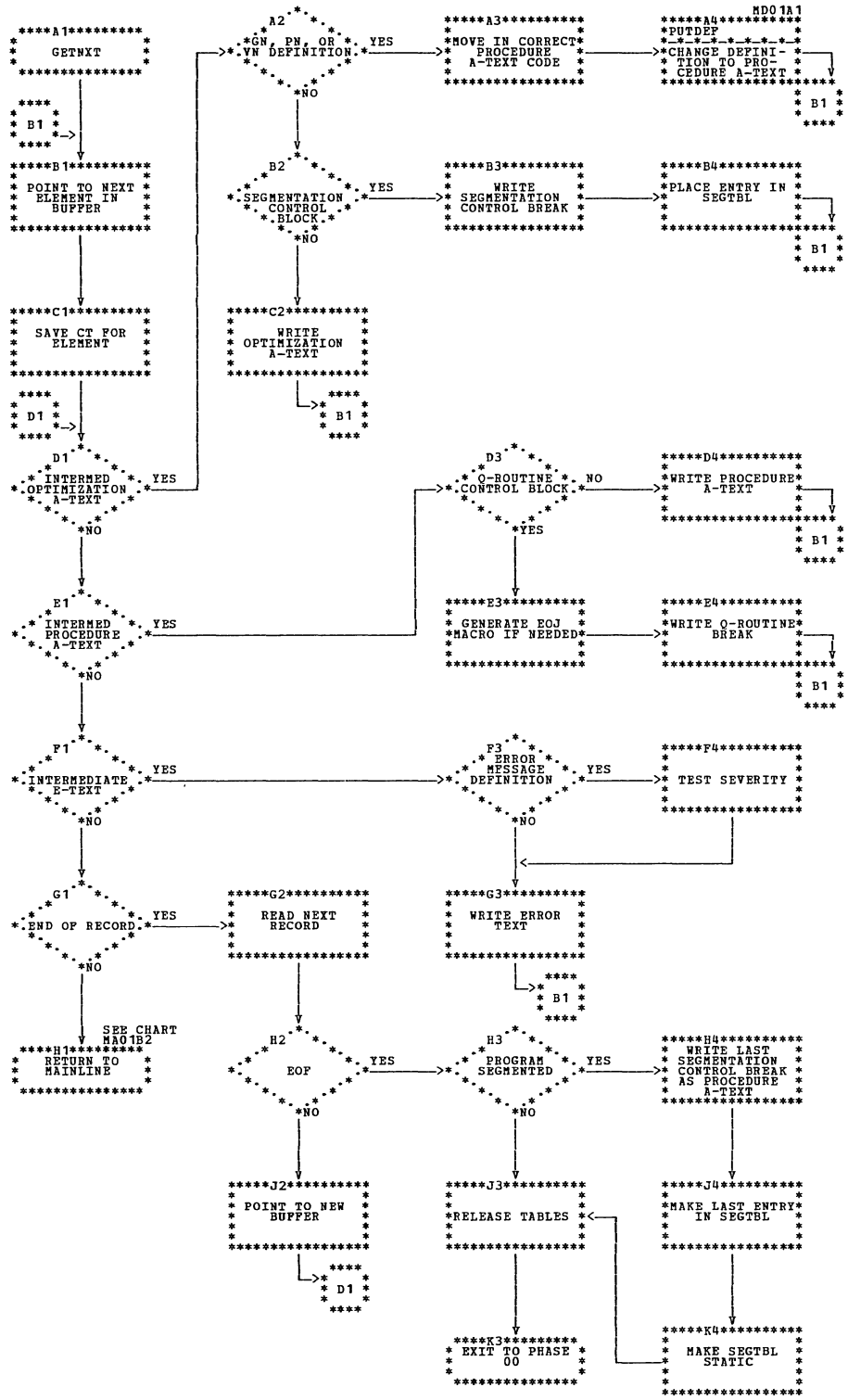


Chart MD. Phase 51: PUTDEF Routine

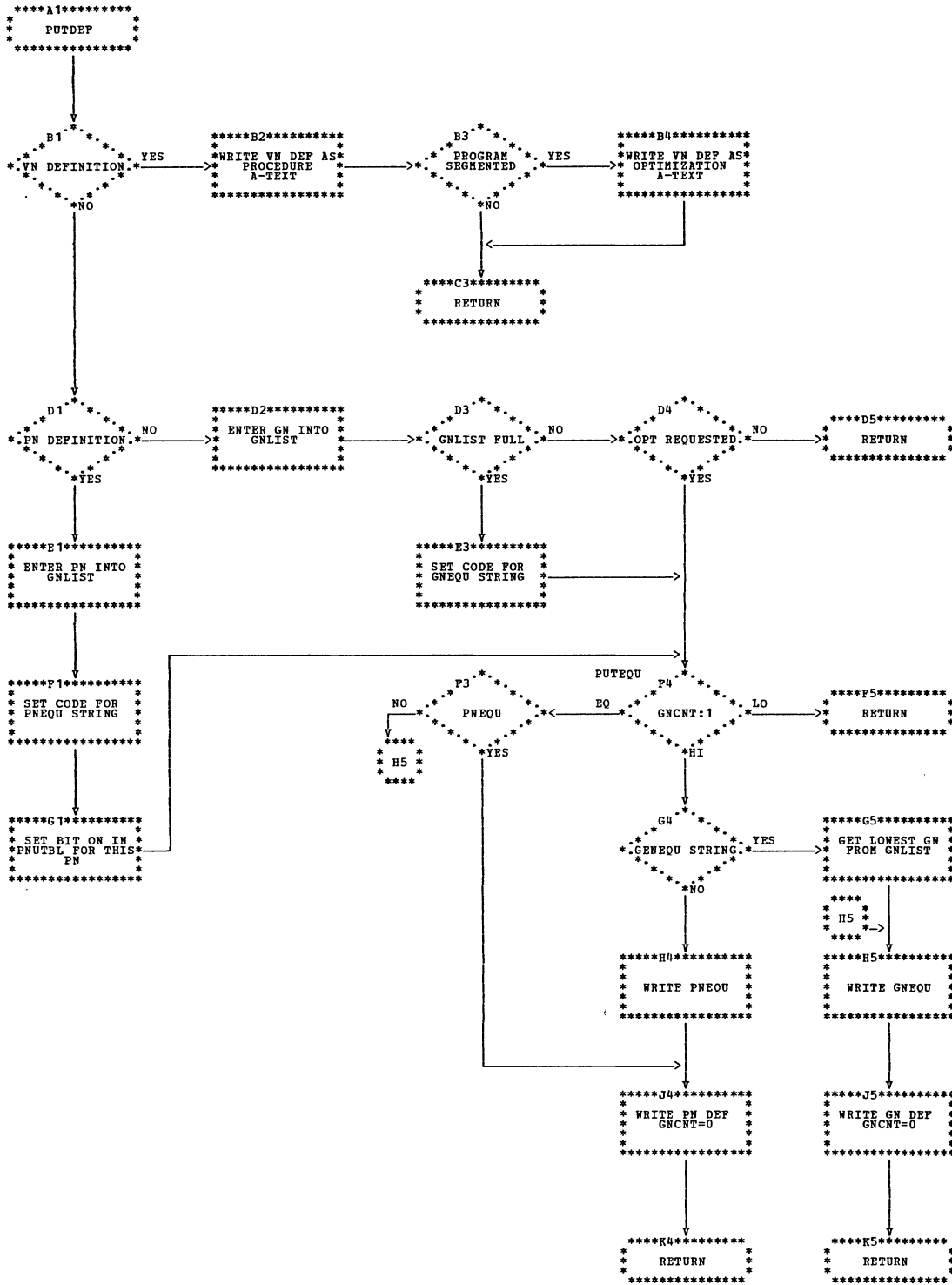


Chart ME. Phase 51: SET Verb Analyzer, Format 1

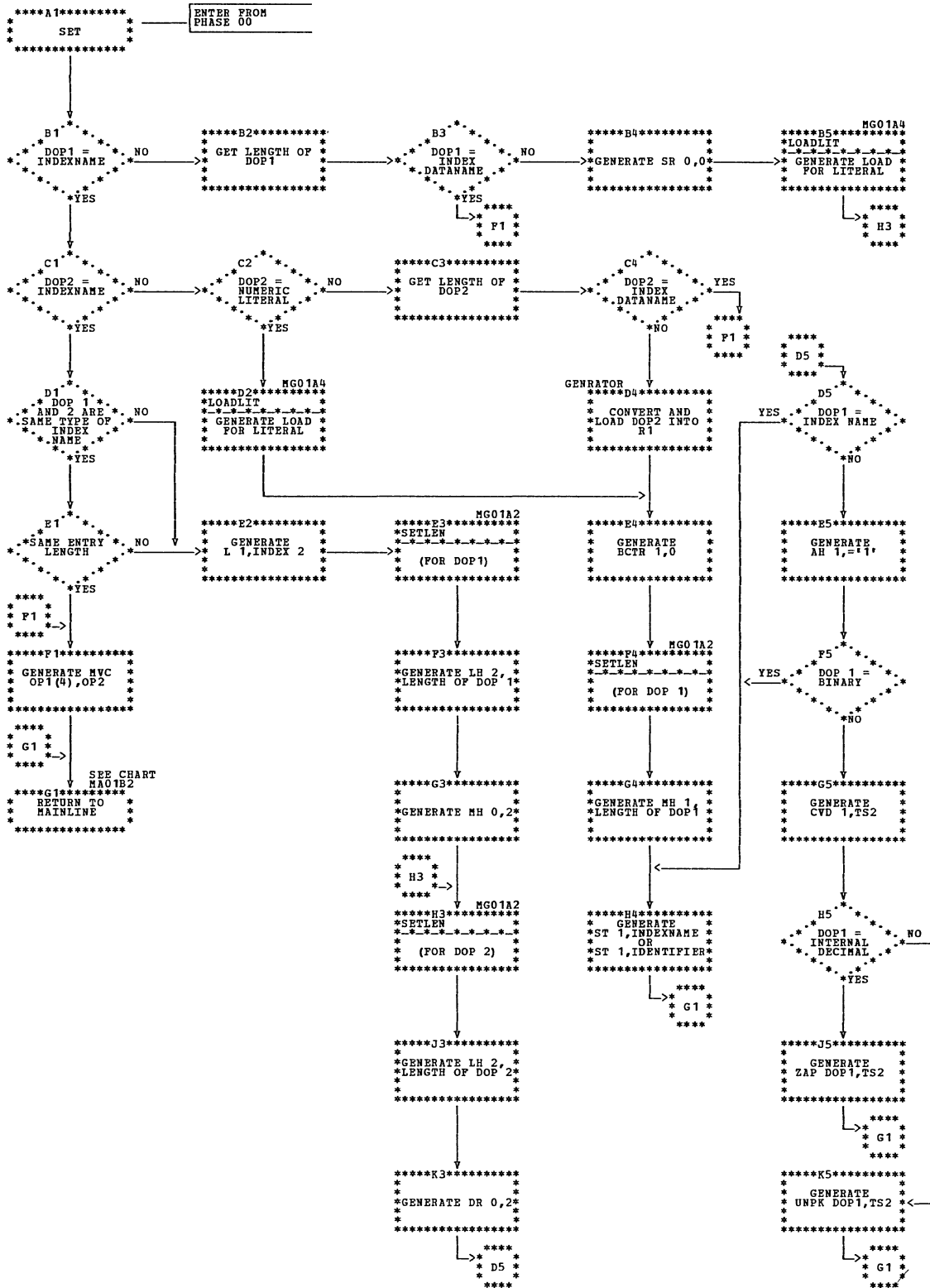


Chart MF. Phase 51: MOVE4

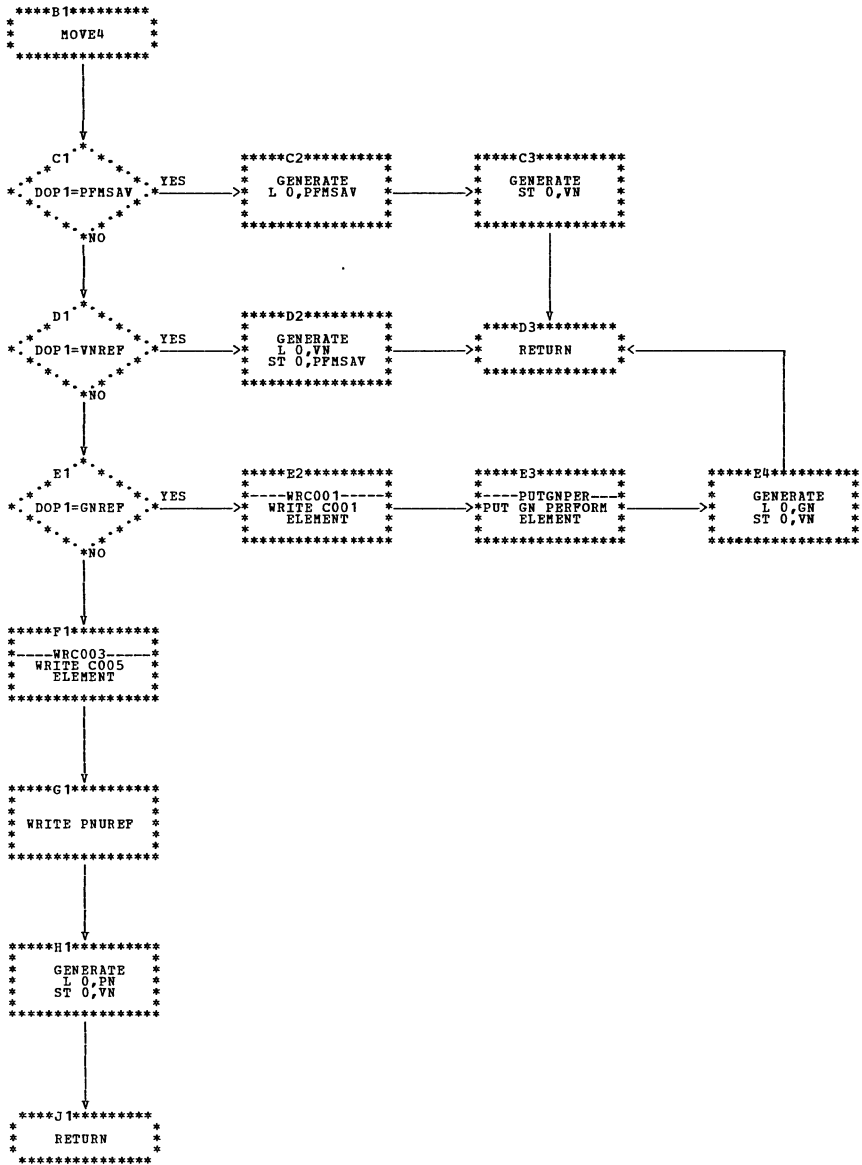


Chart MG. Phase 51: SETLEN and LOADLIT

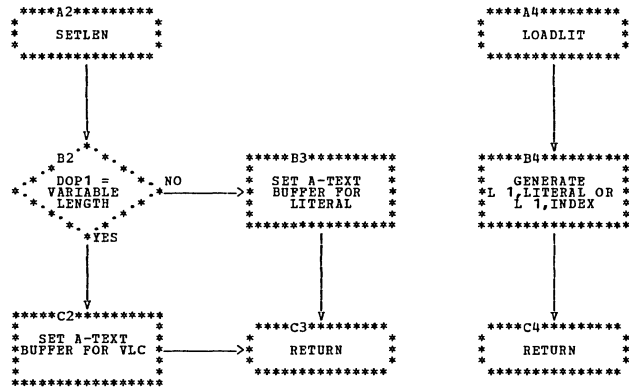


Chart MH. Phase 51: SET Verb Analyzer, Format 2

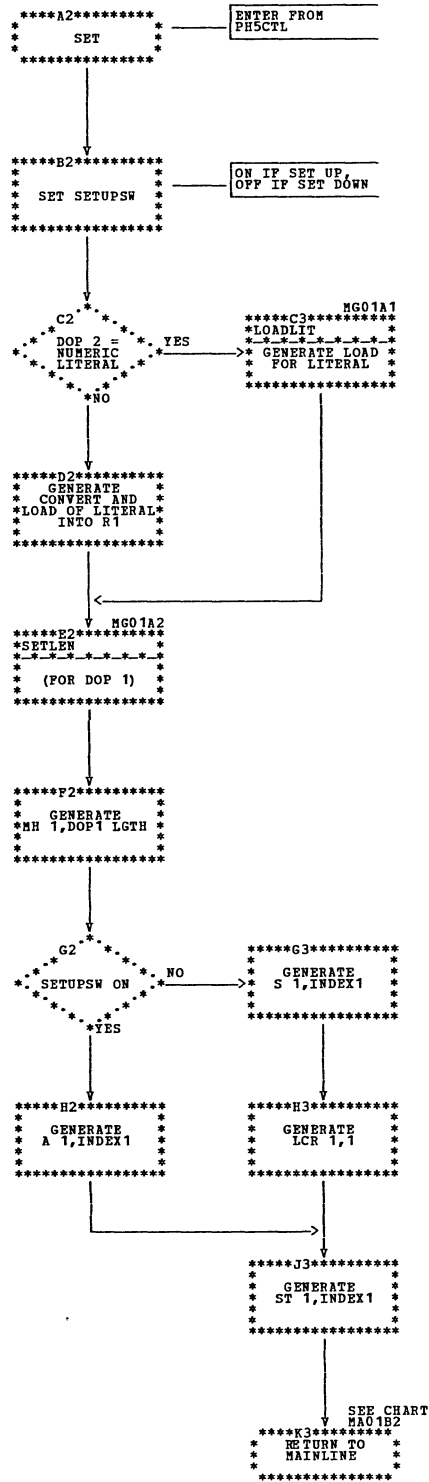




Chart MI. Phase 51: PERFORM and TRANSFORM

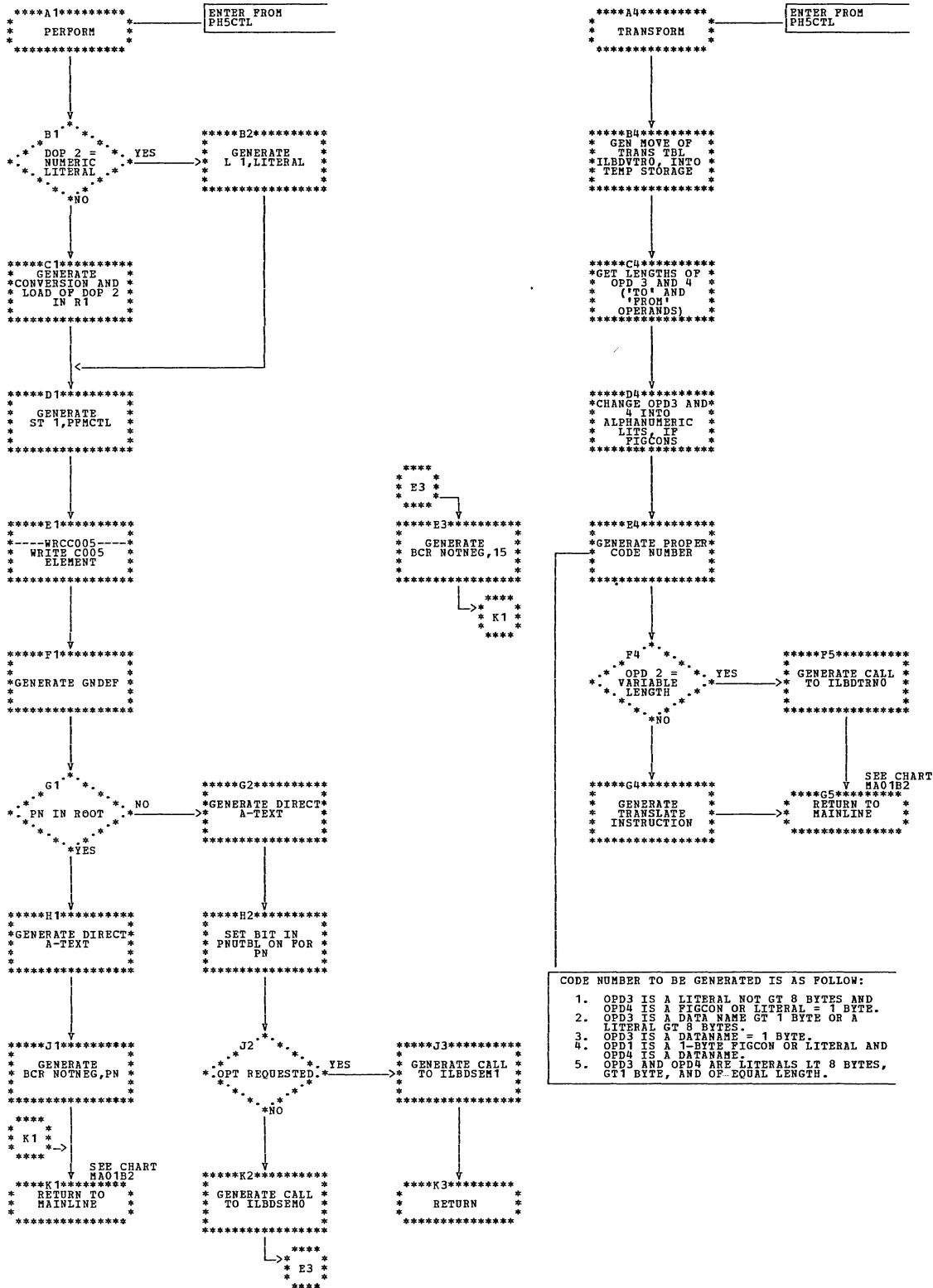


Chart MJ. Phase 51: DISPLAY and EQUATE

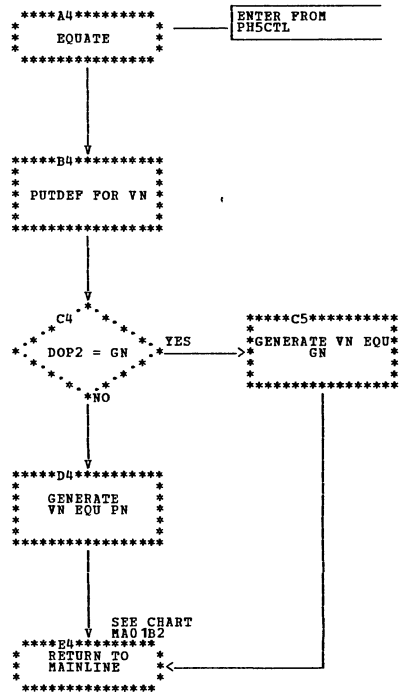
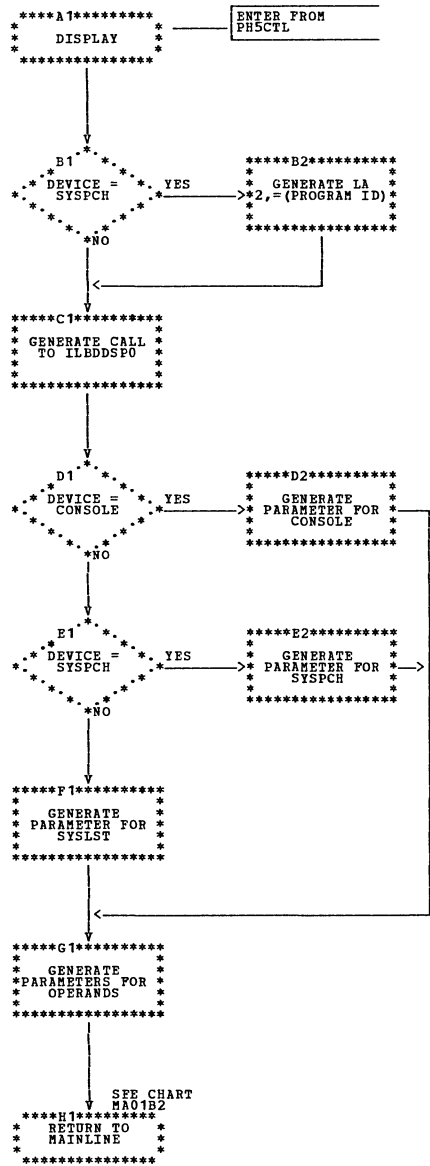




Chart ML. Phase 51: DEBUG, READ, TRACE, and GOBACK Routines

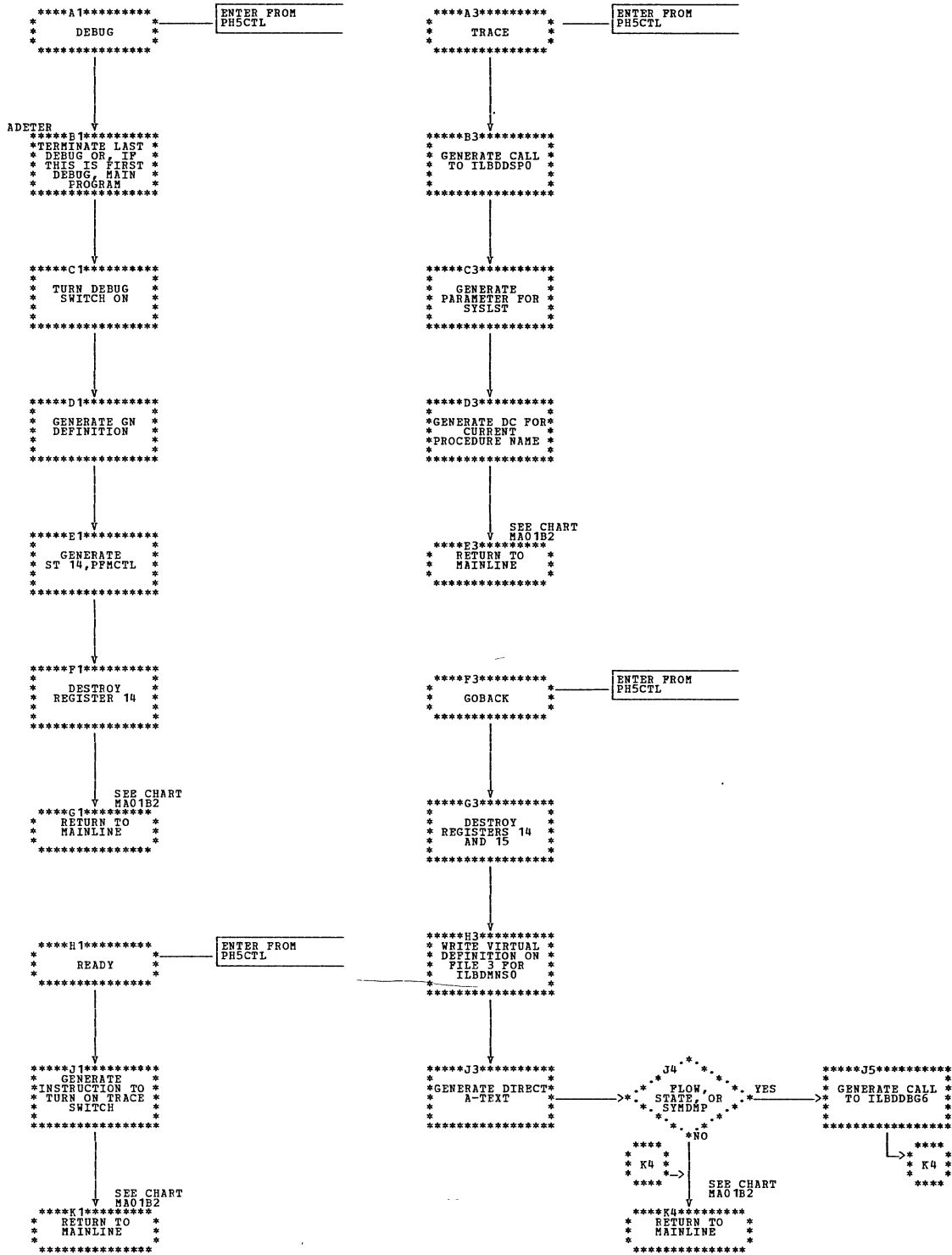


Chart MM. Phase 51: IF-Index Routines

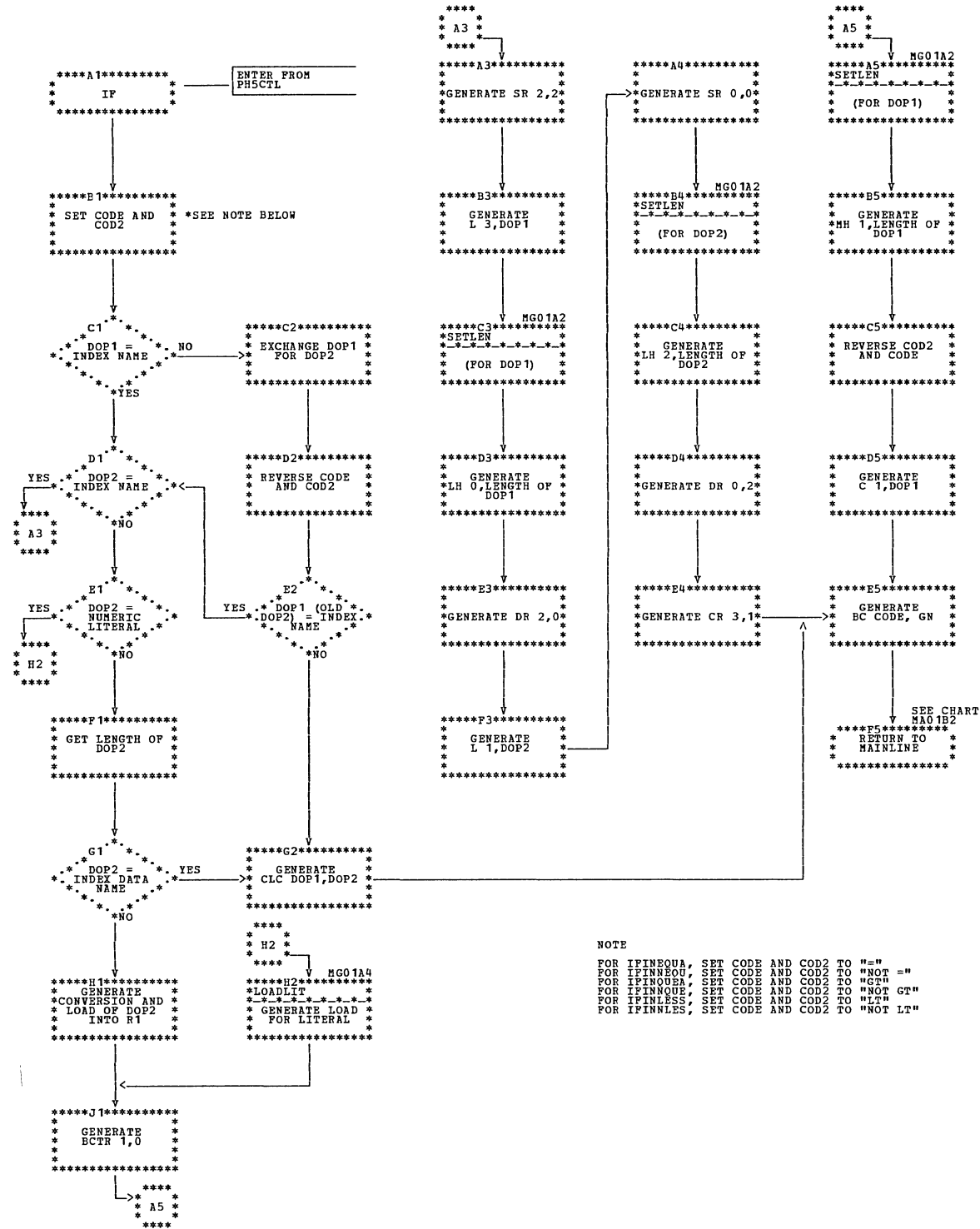


Chart MN. Phase 51: GO, and GODEPM

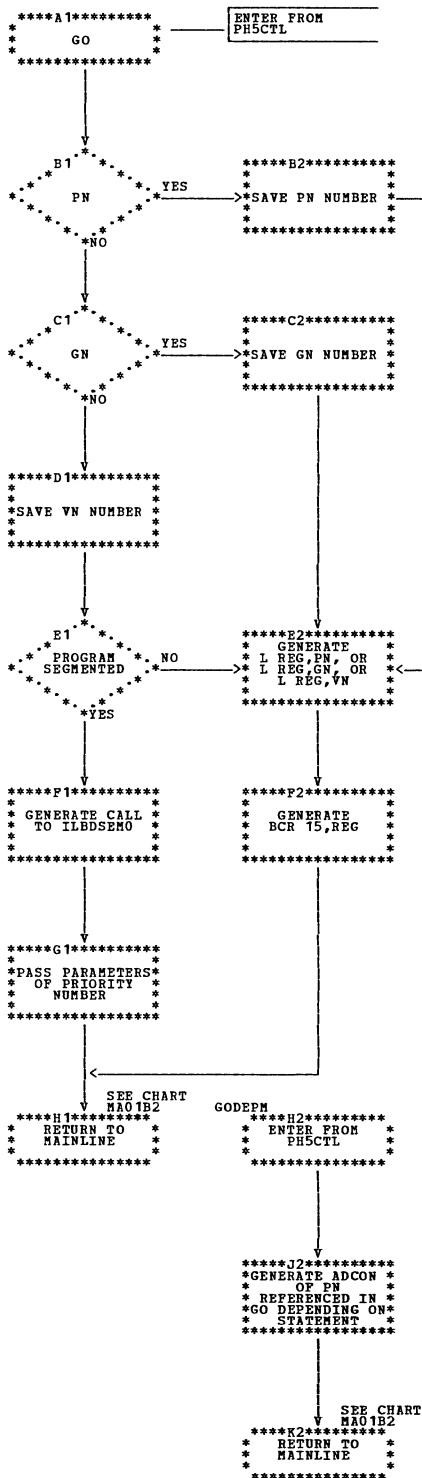


Chart M0. Phase 51: GODEPL and GO DEPENDING

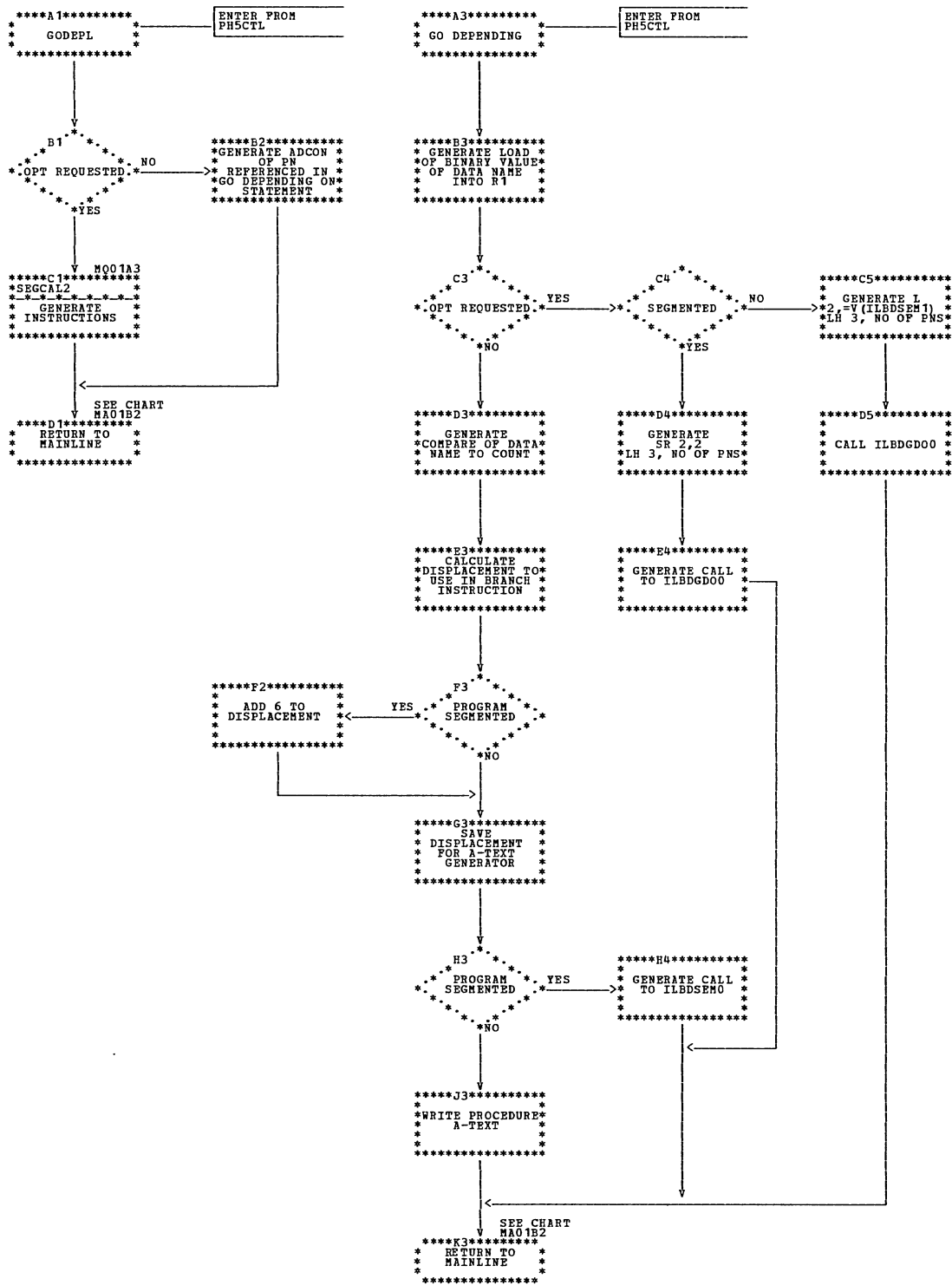


Chart MP. Phase 51: Nonnumeric IF (IFANAL) and Class Test (CLANAB) Processors

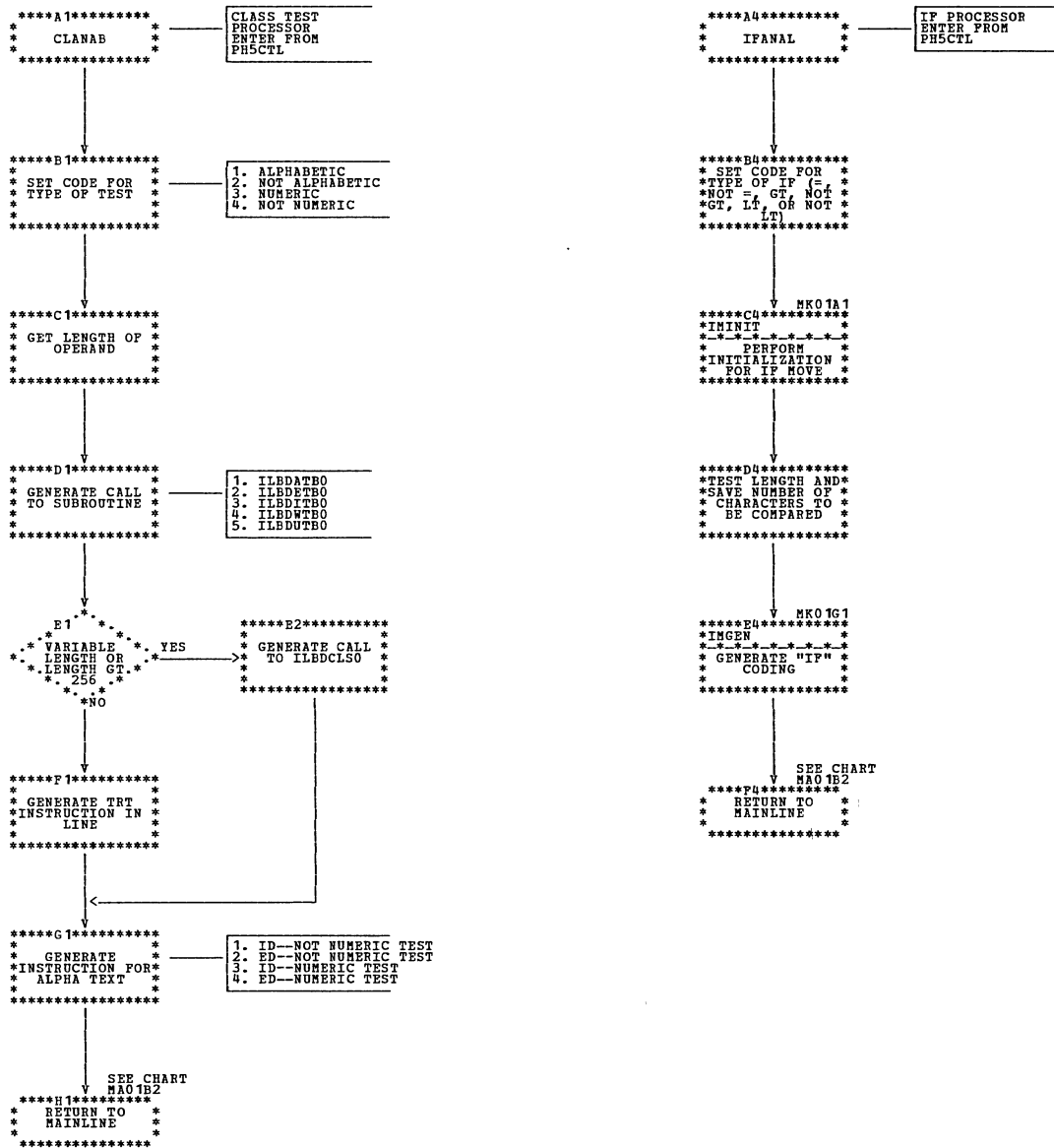




Chart HQ. Phase 51: SEGAL and SEGCAL3

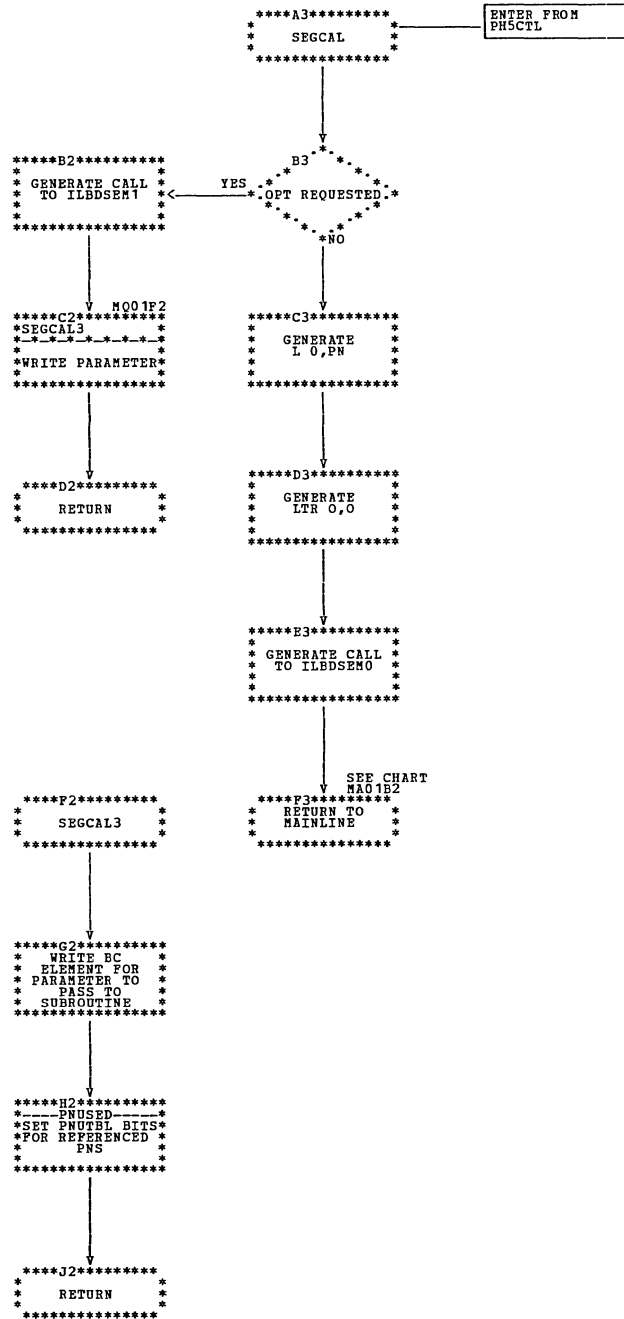


Chart MR. Phase 51: A-text Generator, and GATXTC and GATXTV Routines

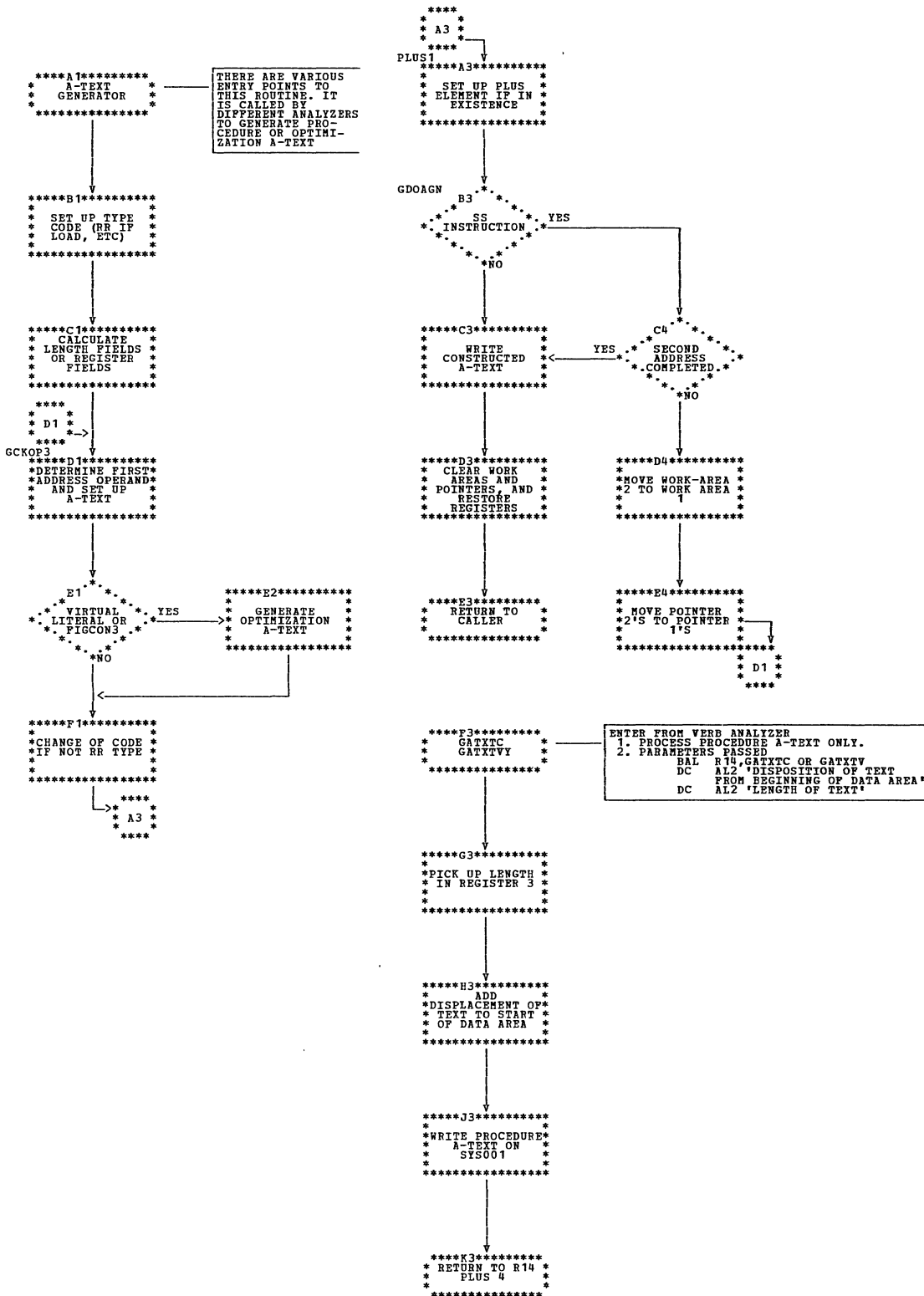


Chart NA. Phase 60 (ILACEL60): Overall Logic

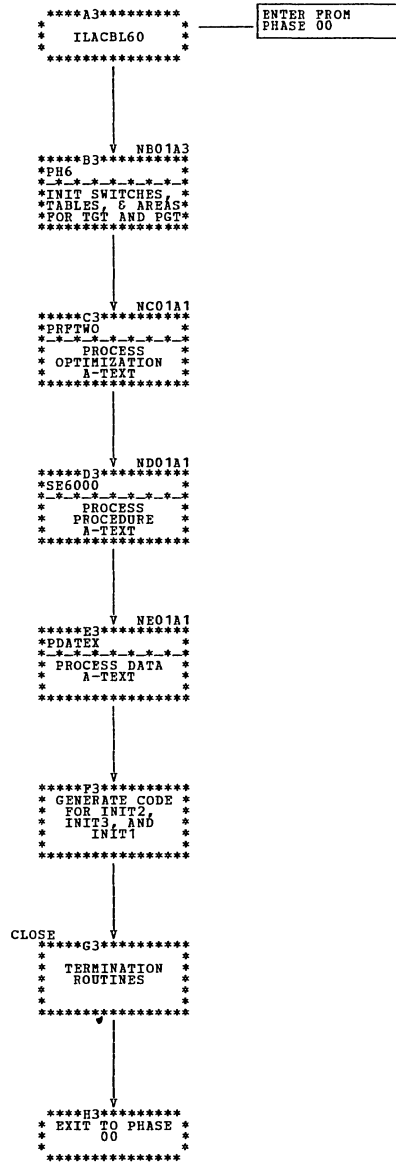


Chart NB. Phase 60: PH6 Routine

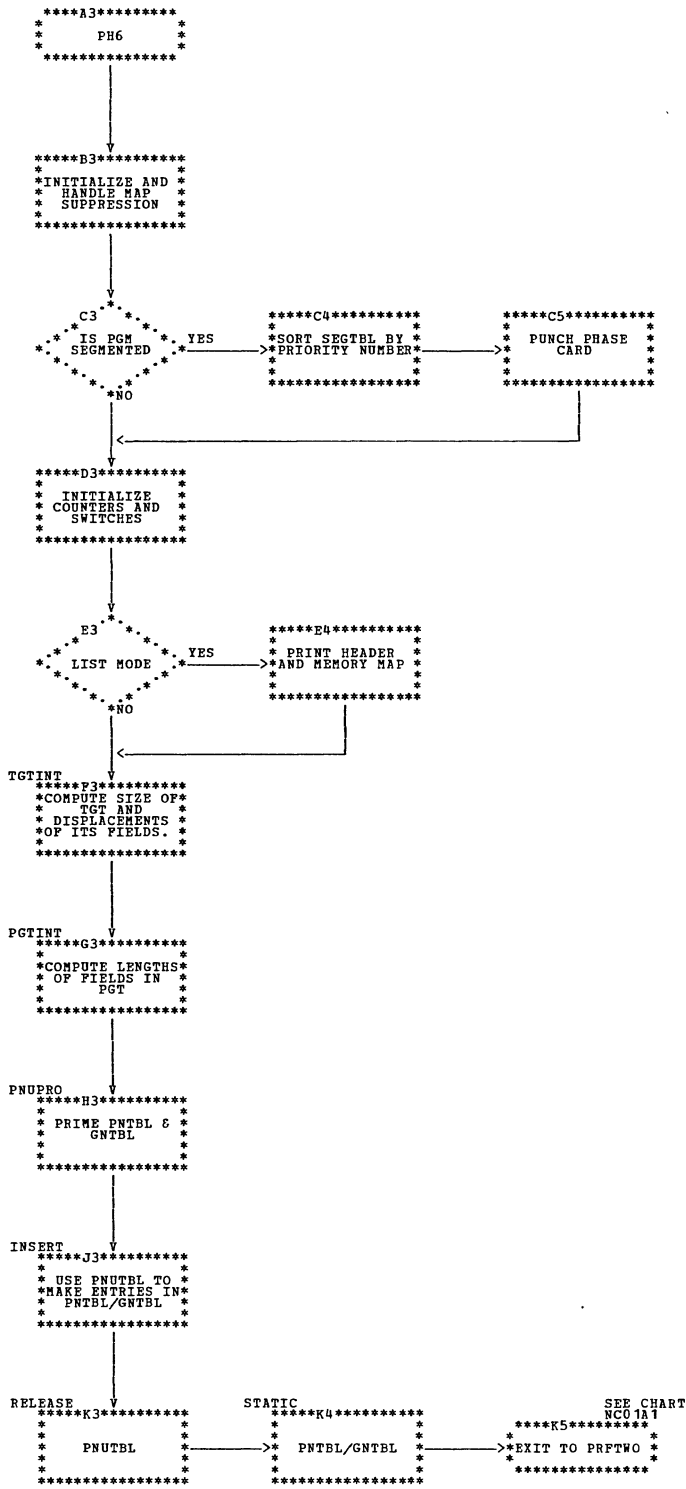


Chart WC. Phase 60: PRFTWO Routine

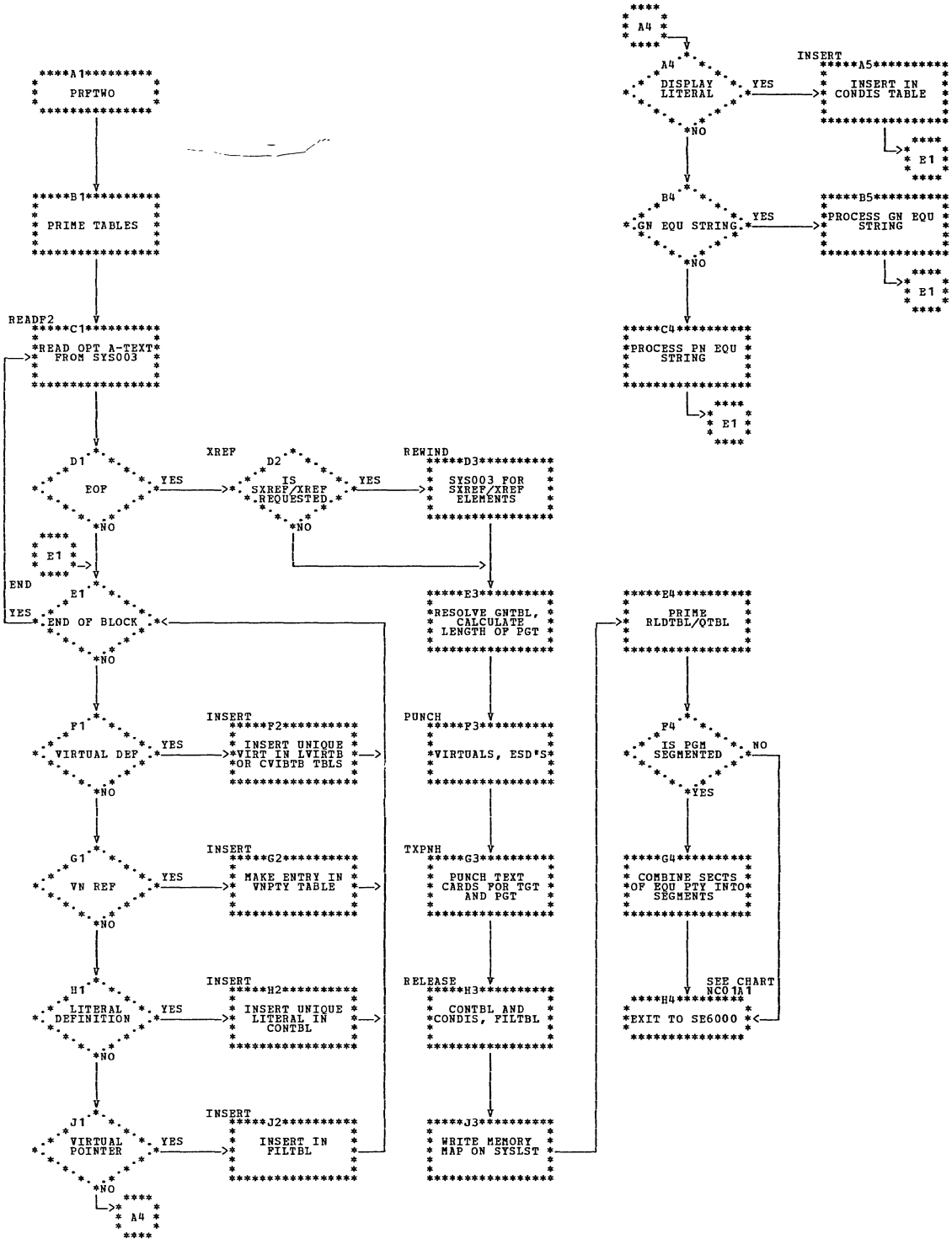


Chart ND. Phase 60: SE6000 Routine

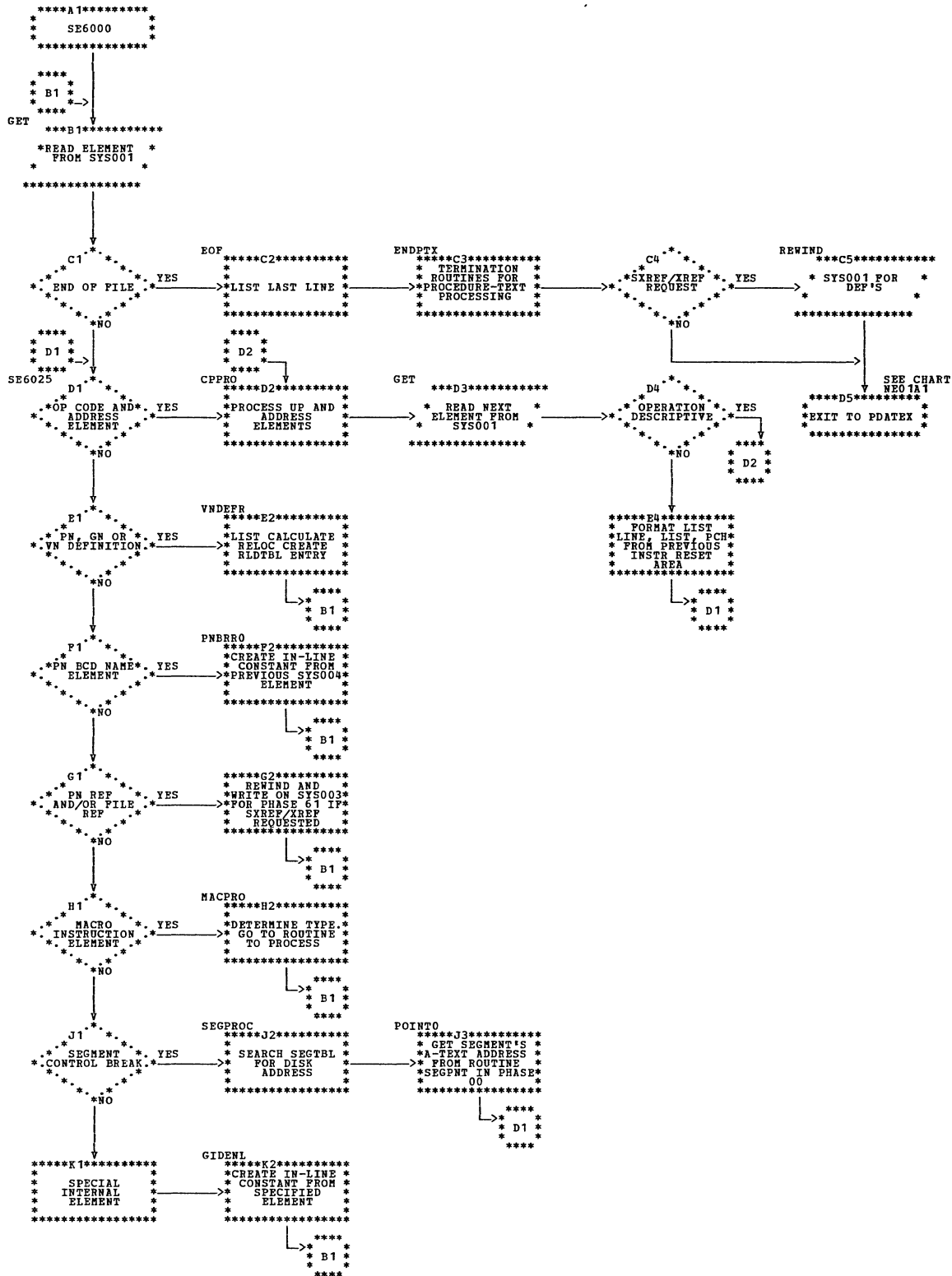


Chart NE. Phase 60: PDATEX (Part 1 of 2)

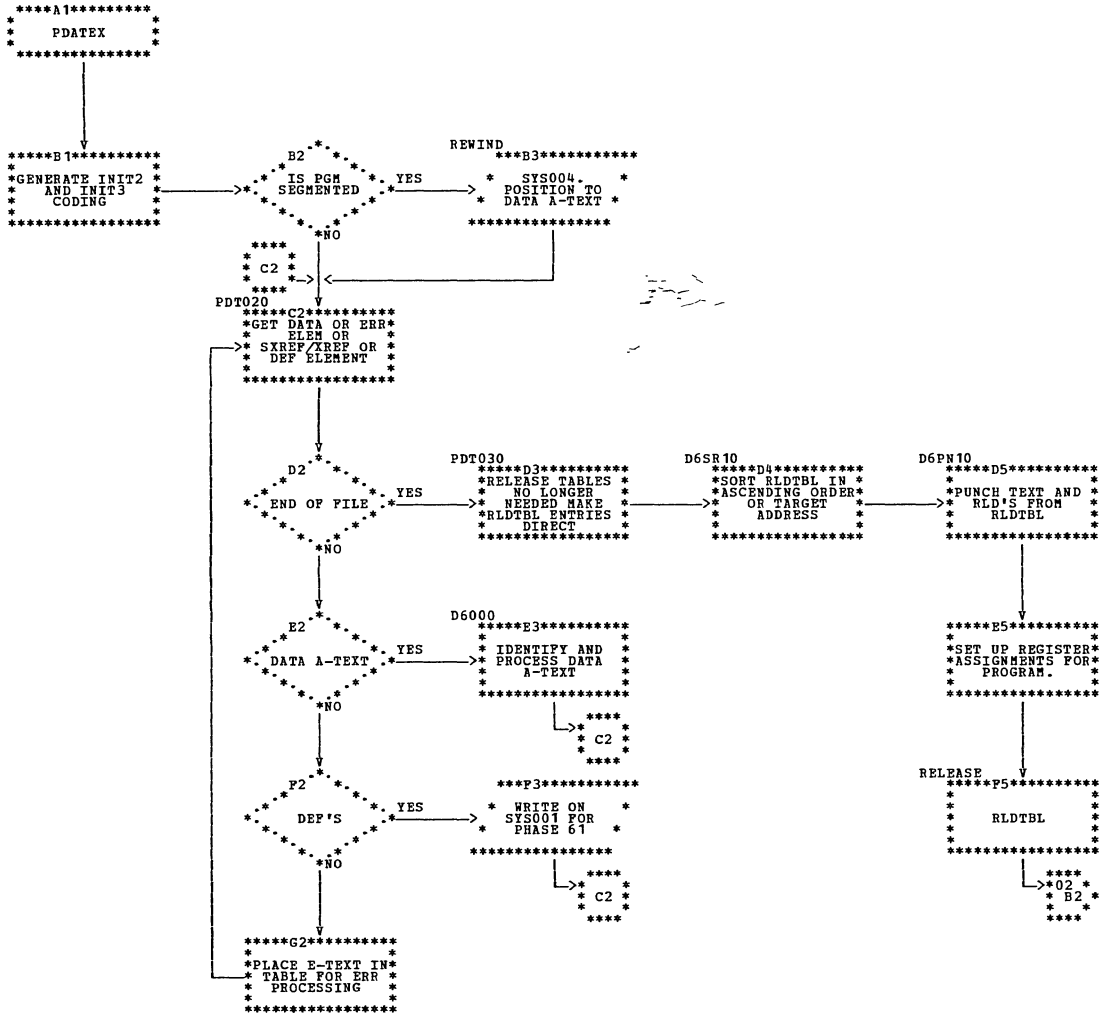


Chart NF. Phase 60: PDATEX (Part 2 of 2)

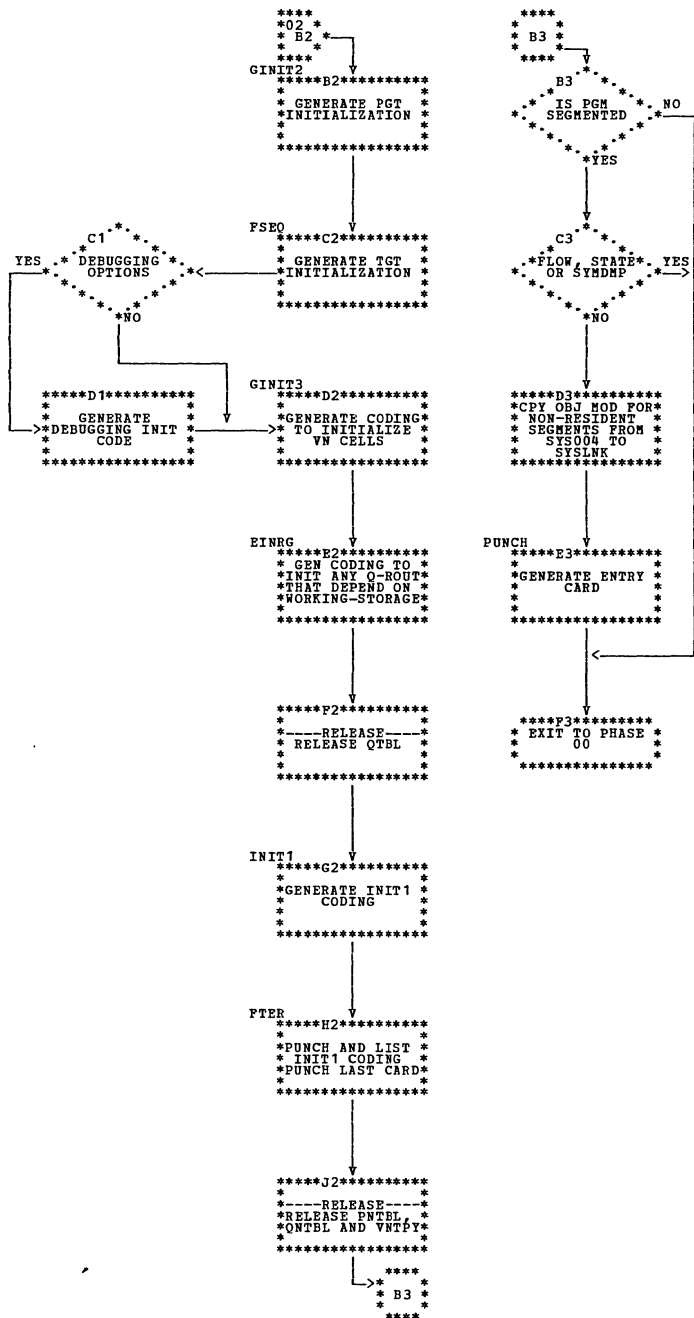




Chart 0A. Phase 62: Overall Logic

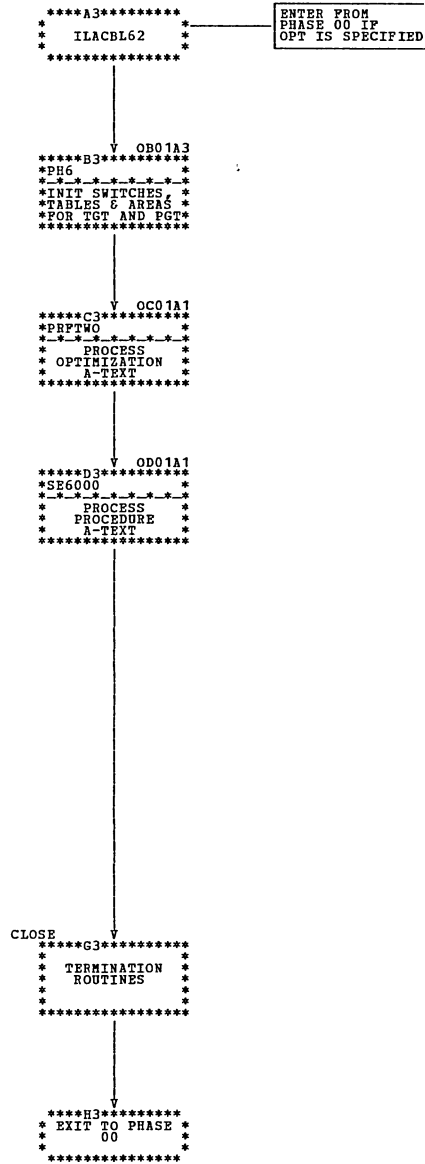


Chart OB. Phase 62: PH6

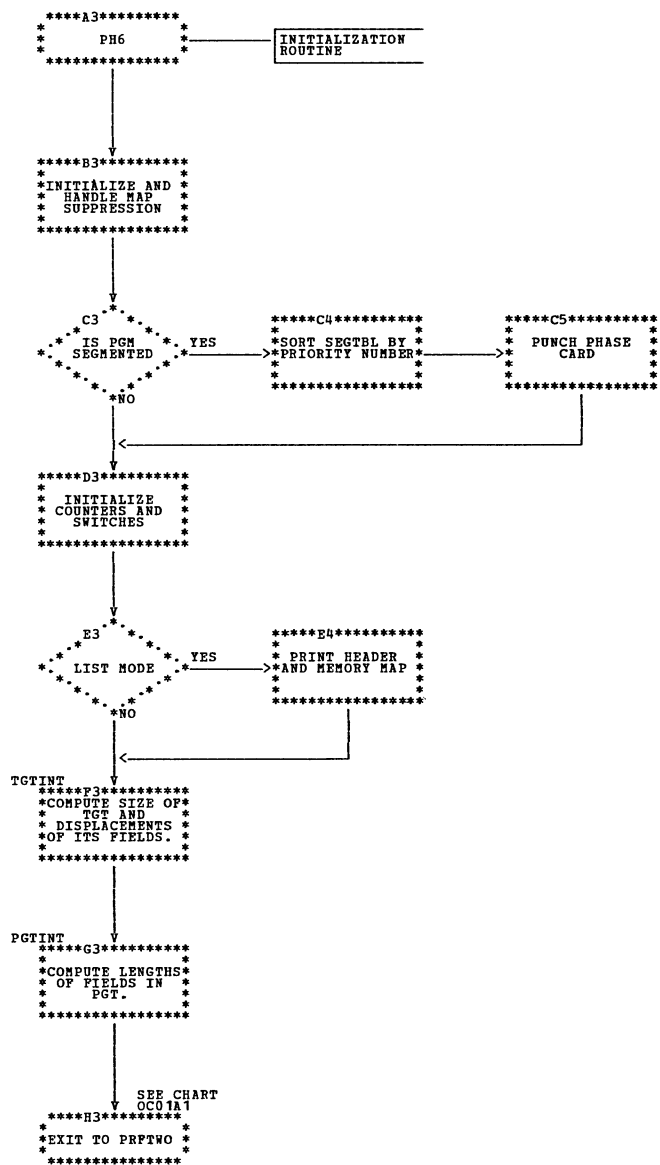


Chart OC. Phase 62: PRFTWO

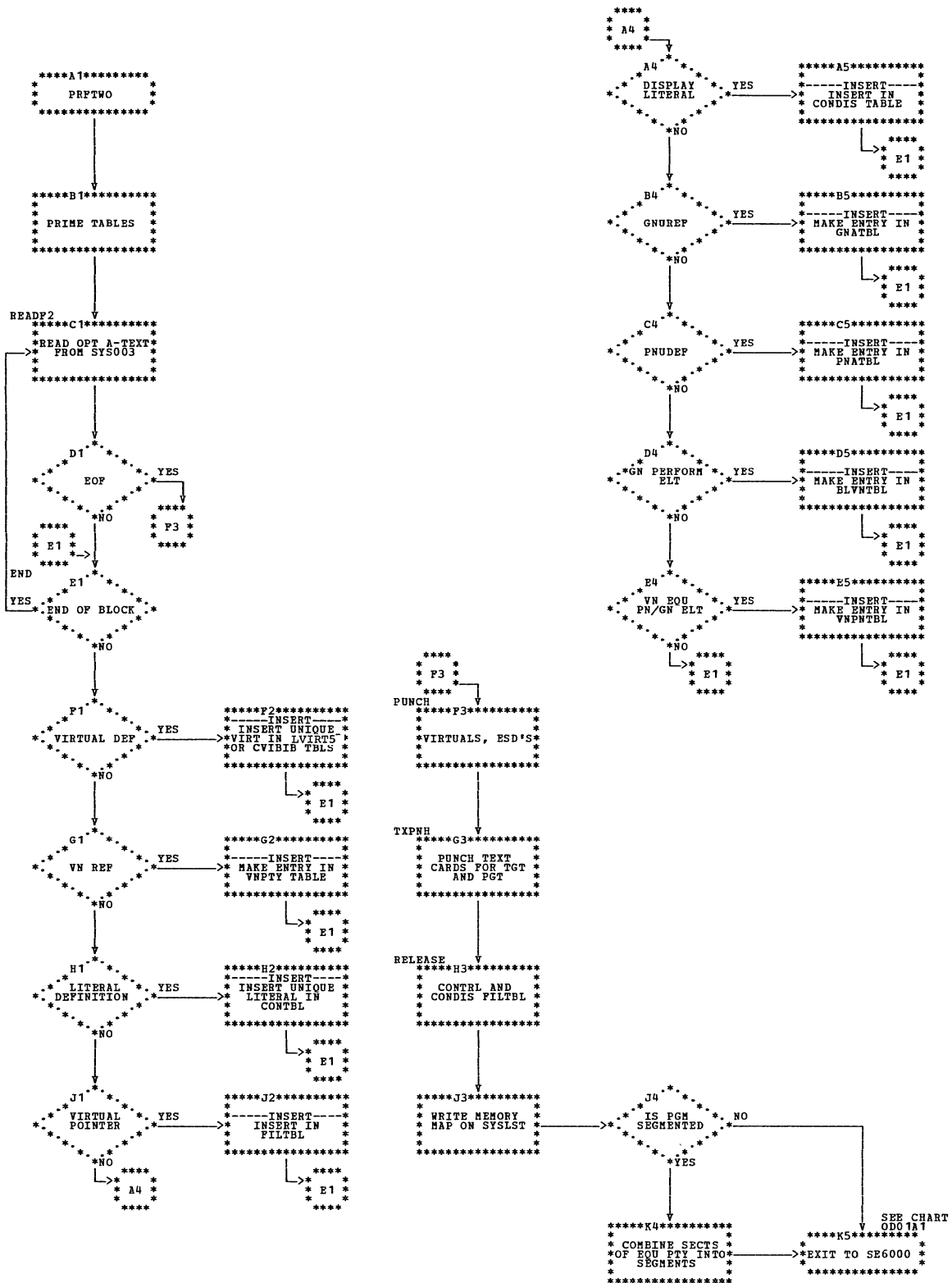




Chart OD. Phase 62: SE6000 (Part 2 of 2)

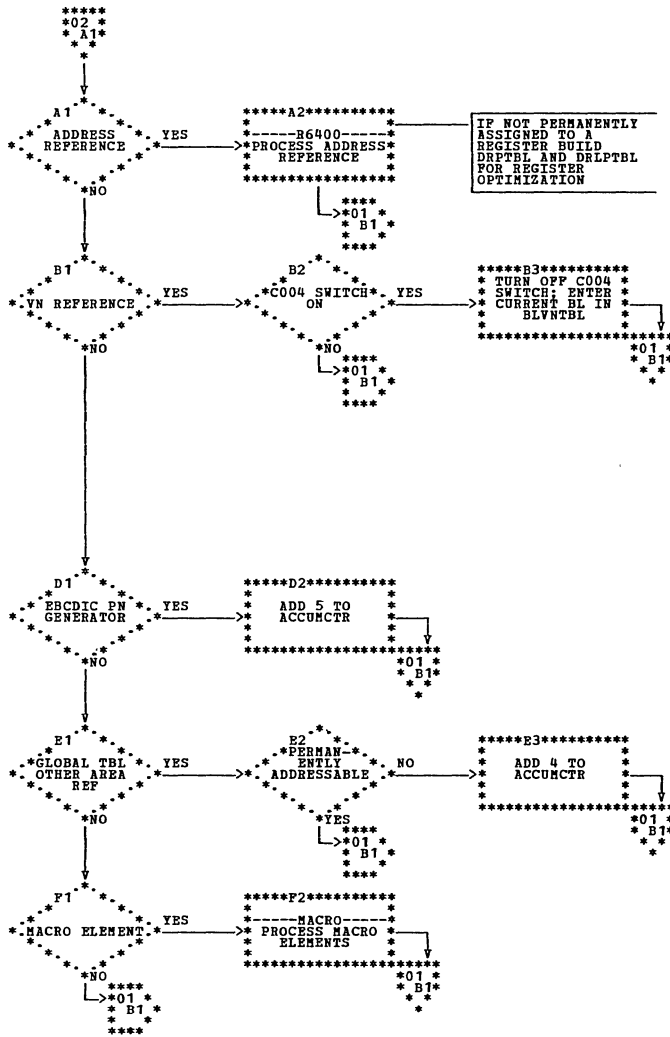


Chart PA. Phase 63 (ILACBL63): Overall Logic

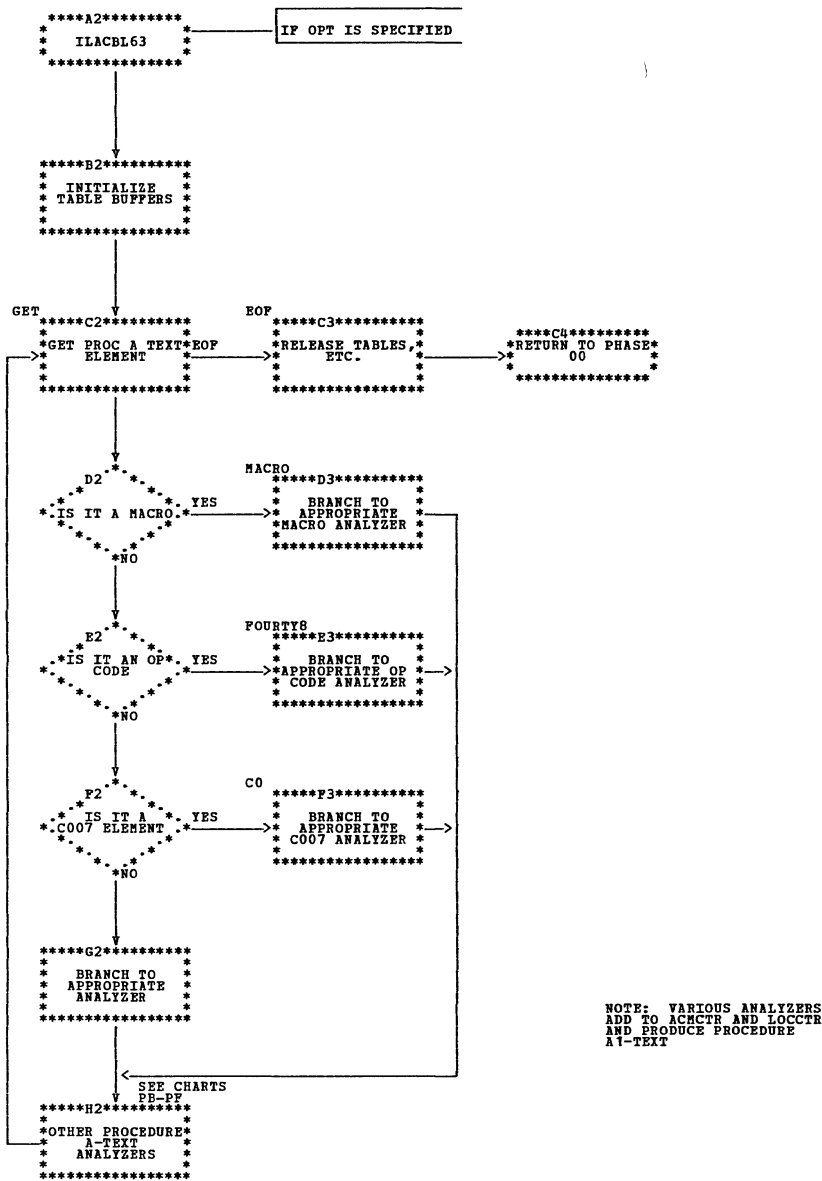


Chart PB. Phase 63: BRANCH

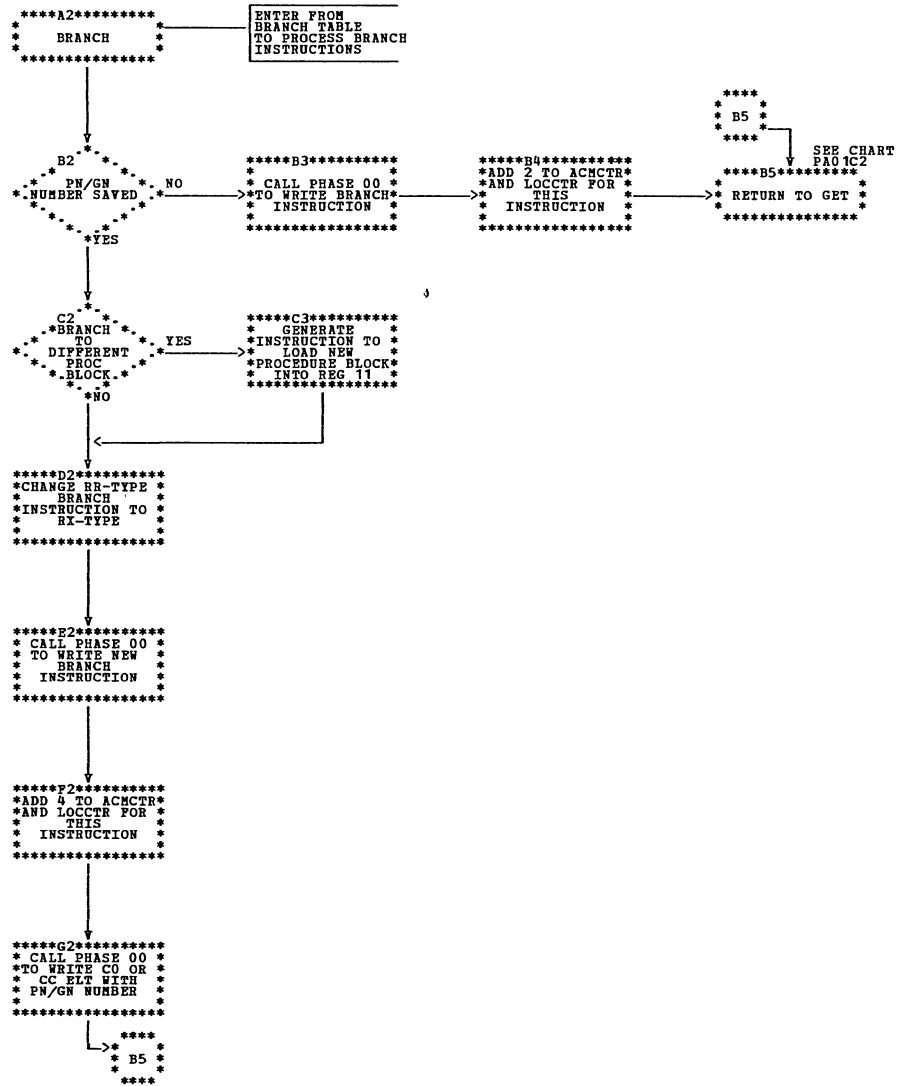


Chart PC. Phase 63: GNDEF

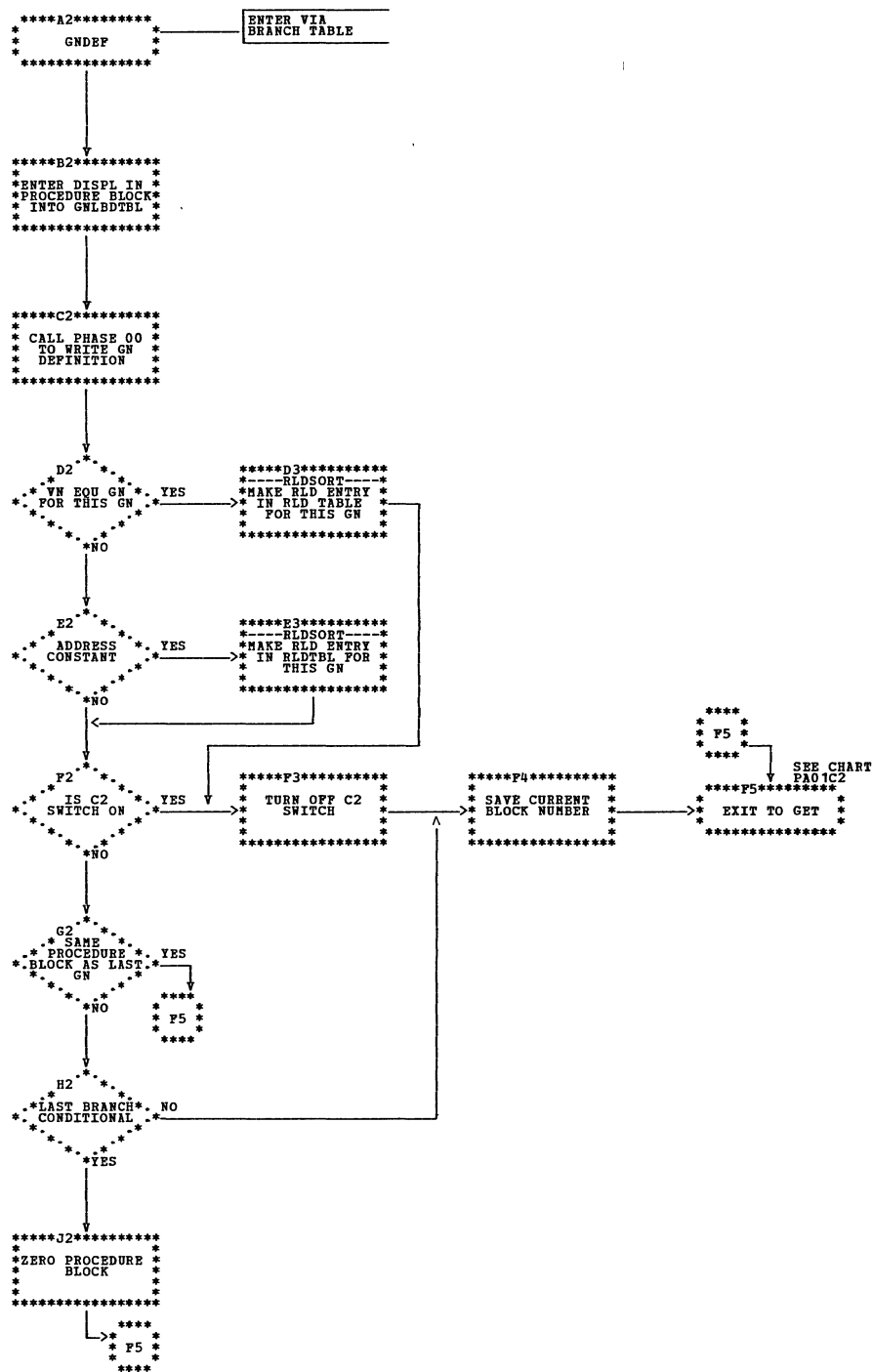




Chart PD. Phase 63: PNDEF

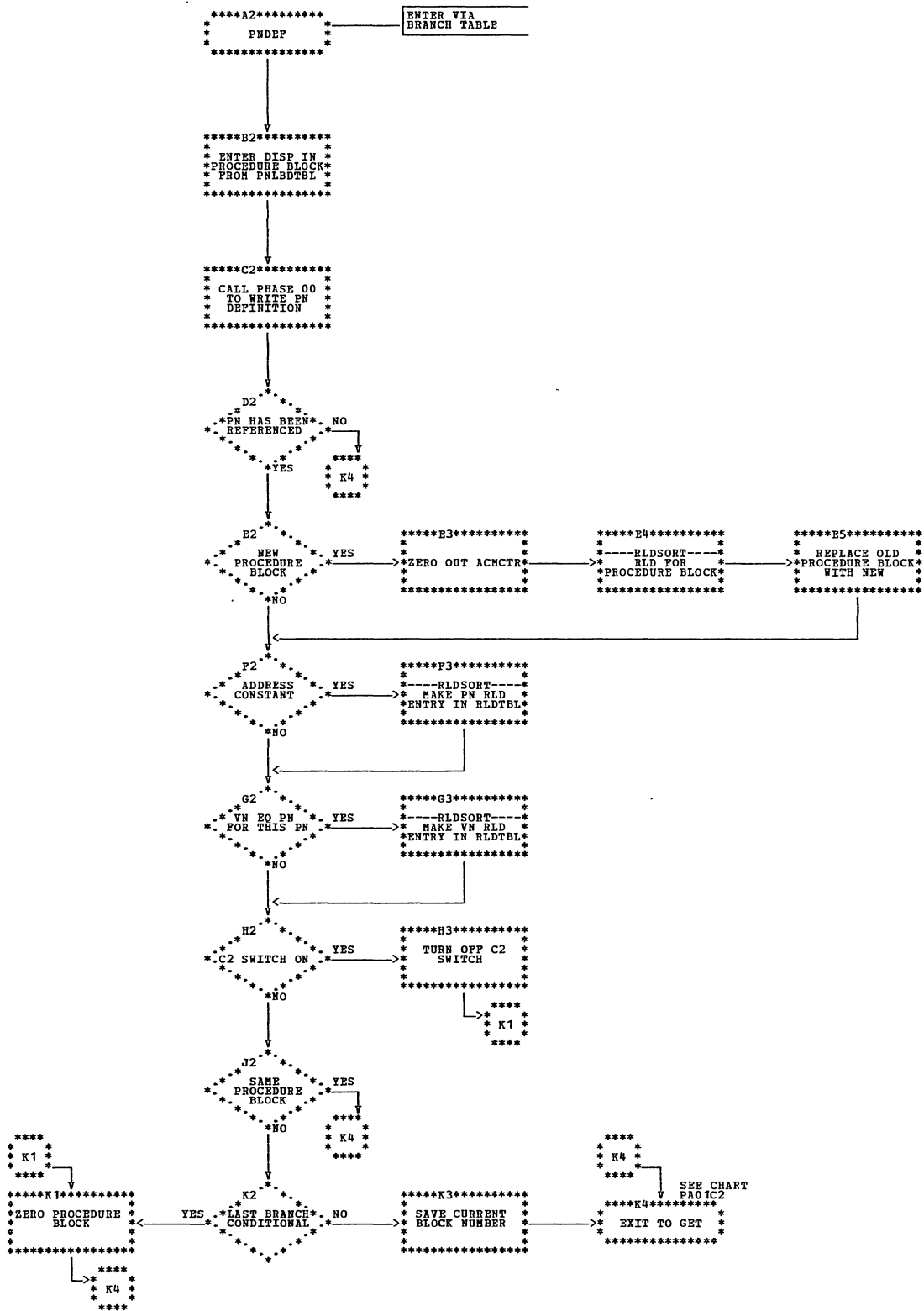


Chart PE. Phase 63: ADREF and ADINCR

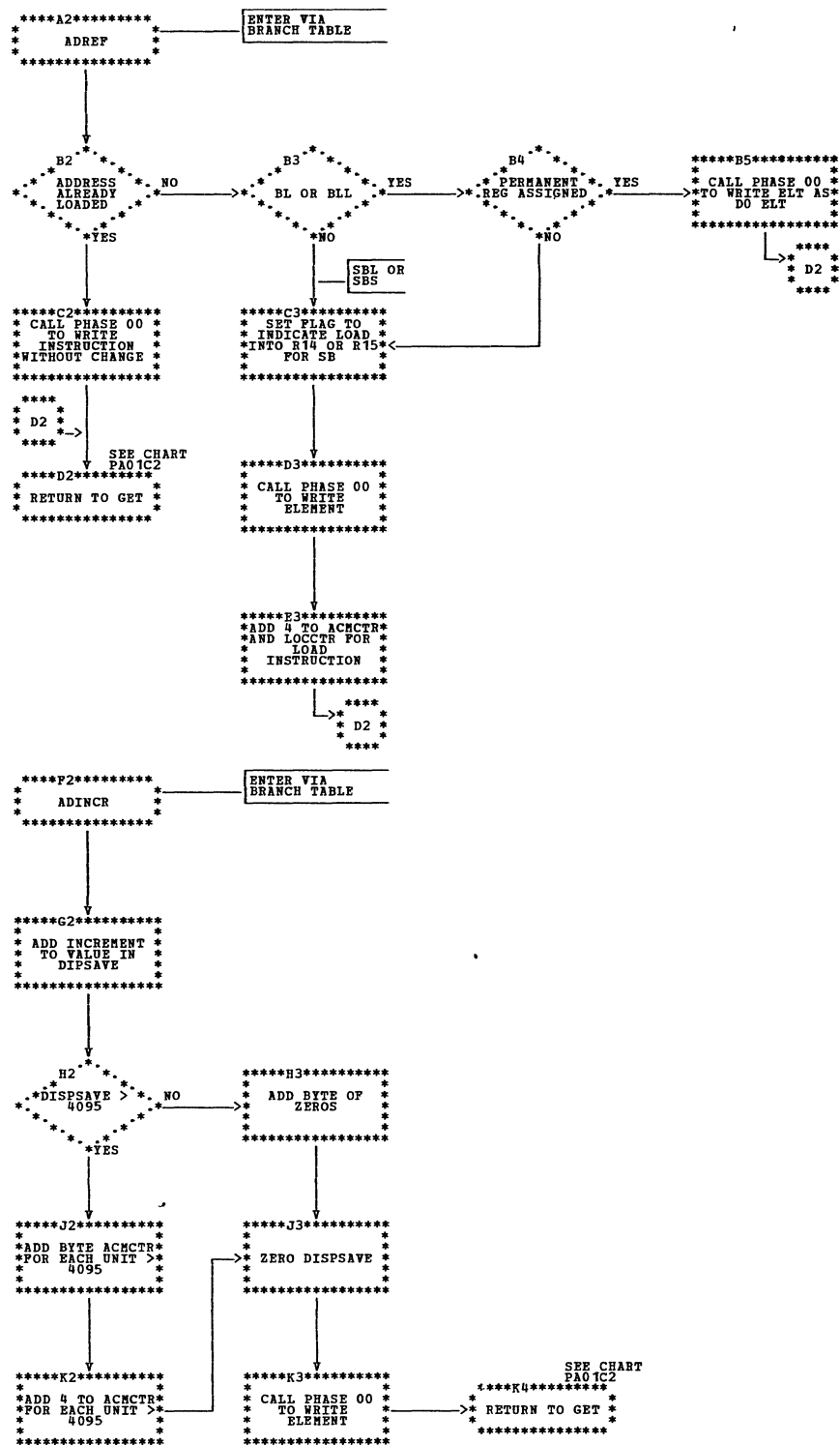


Chart PF. Phase 63: C1REF, PNREF, and GNREF

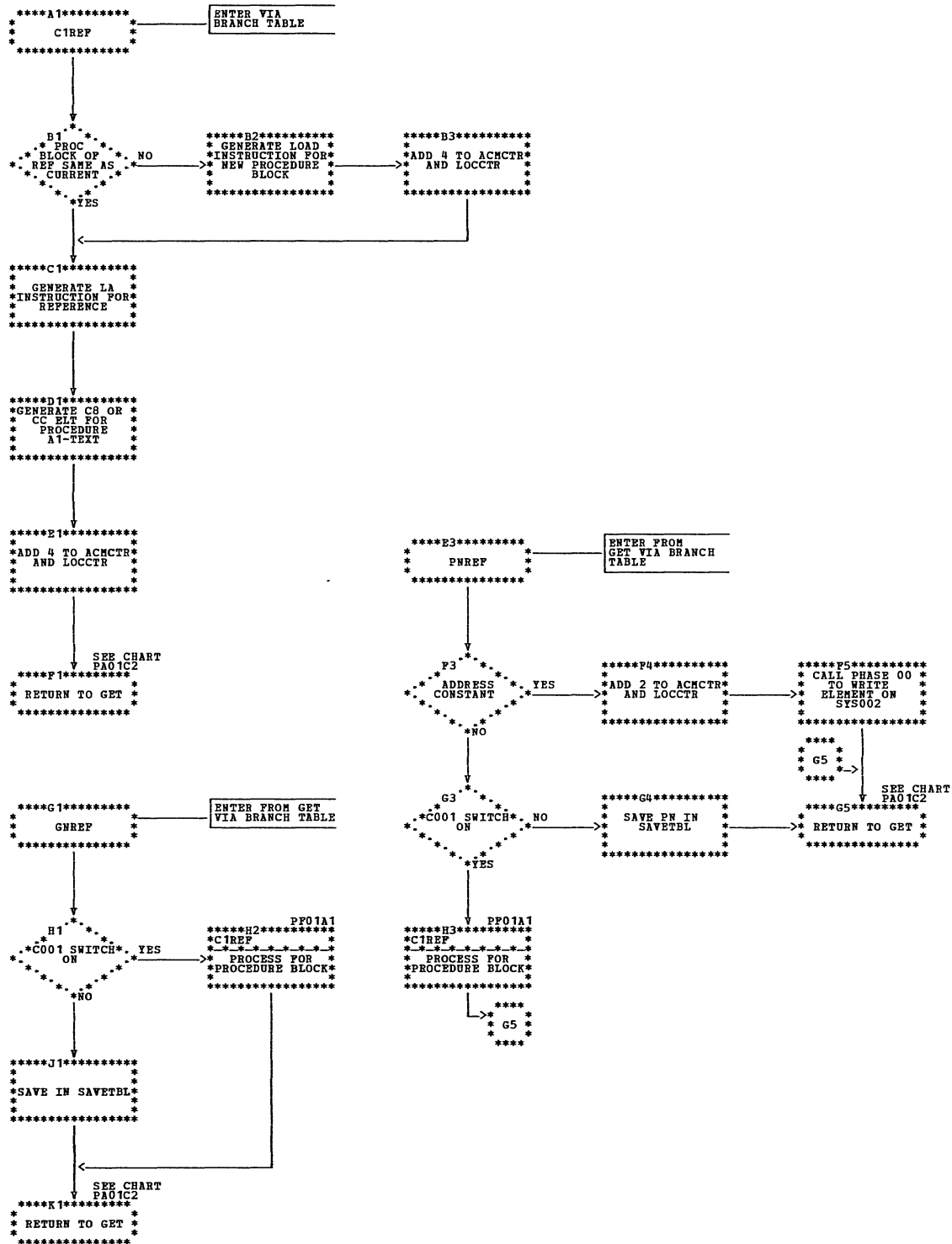


Chart QA. Phase 64 (ILACBL64): Overall Logic

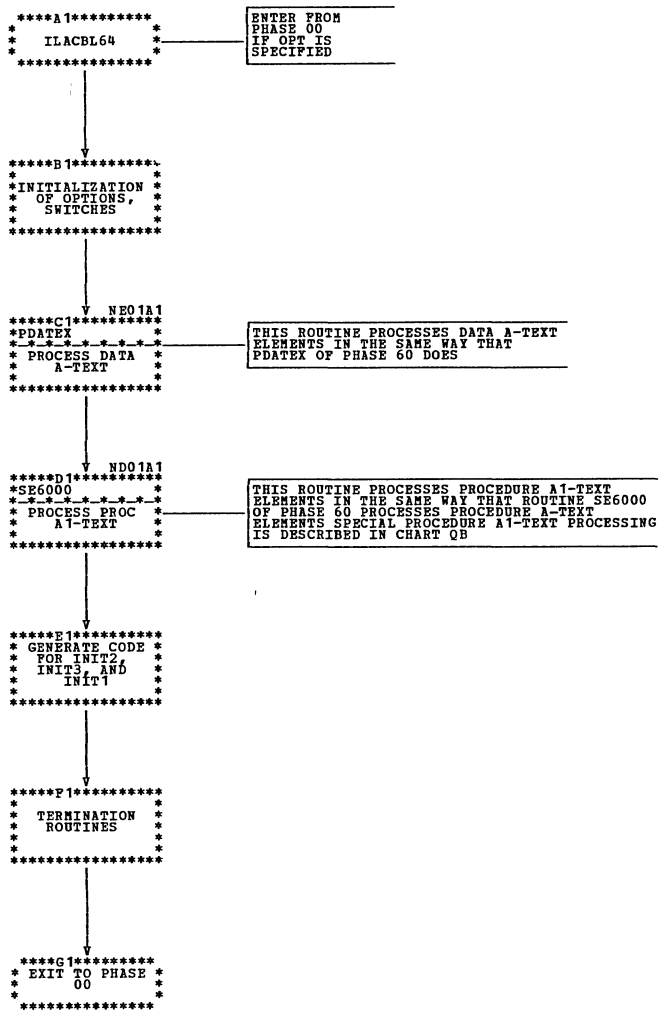


Chart QB. Phase 64: ADREF, RC4, RC8C, and RD001

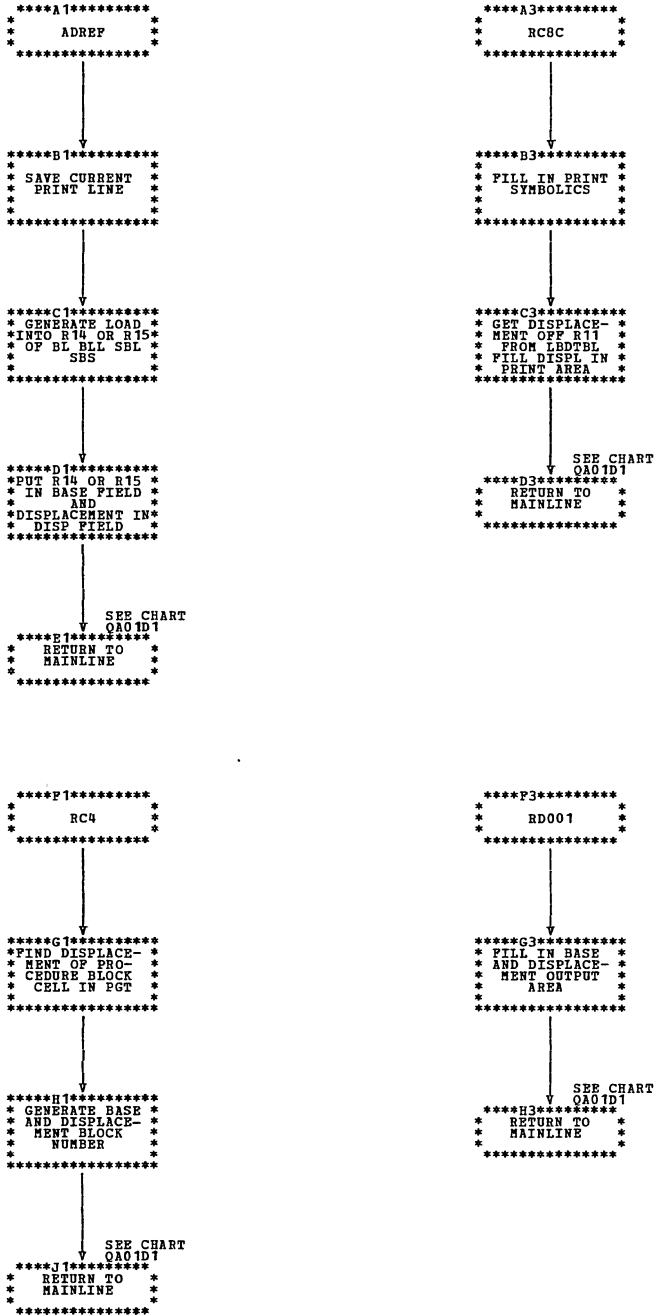


Chart RA. Phase 65 (ILACBL65): Overall Logic

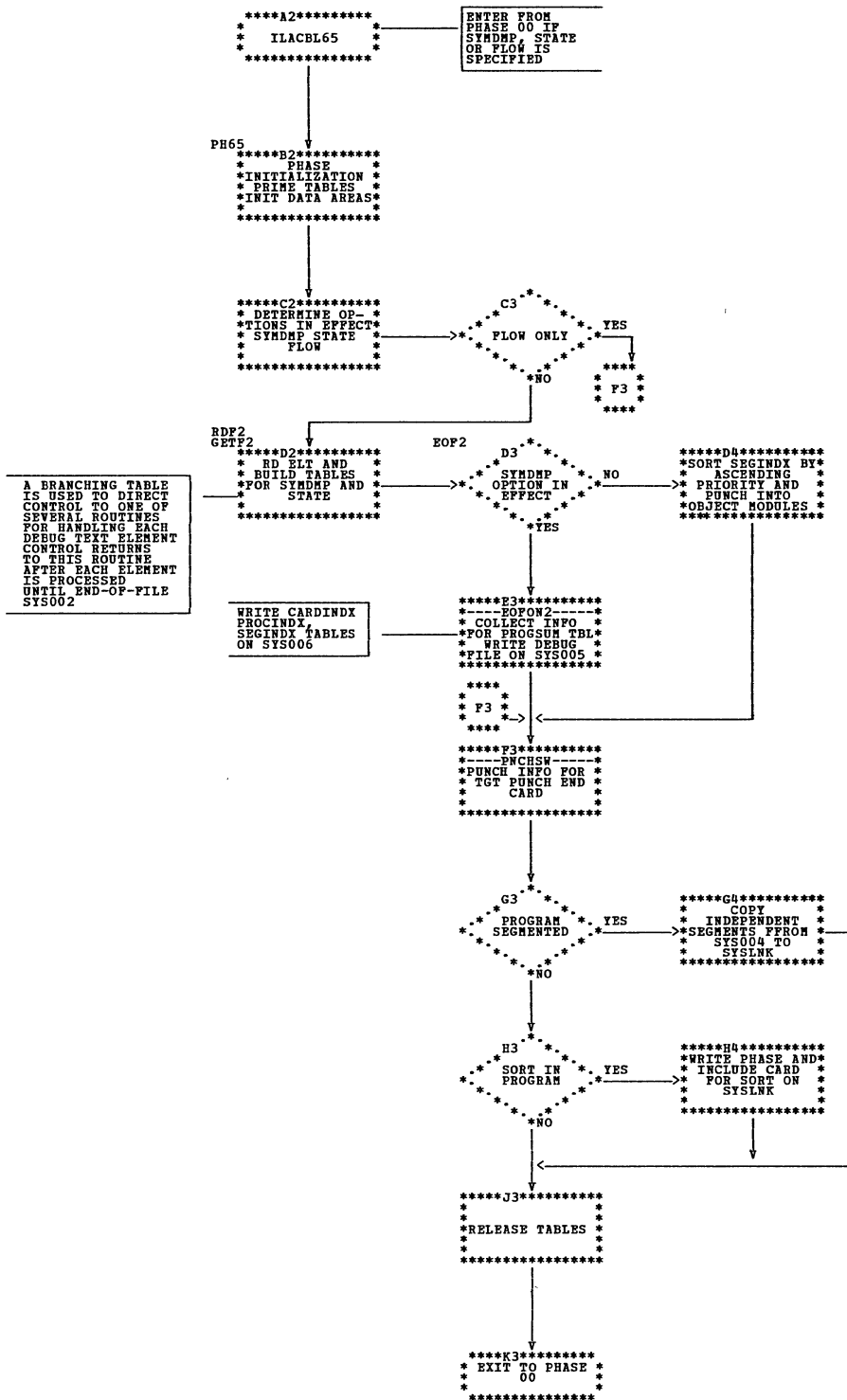


Chart RB. Phase 65: Debug-text Element Processors (TENPROC, TWENPROC, GTEQ10K)

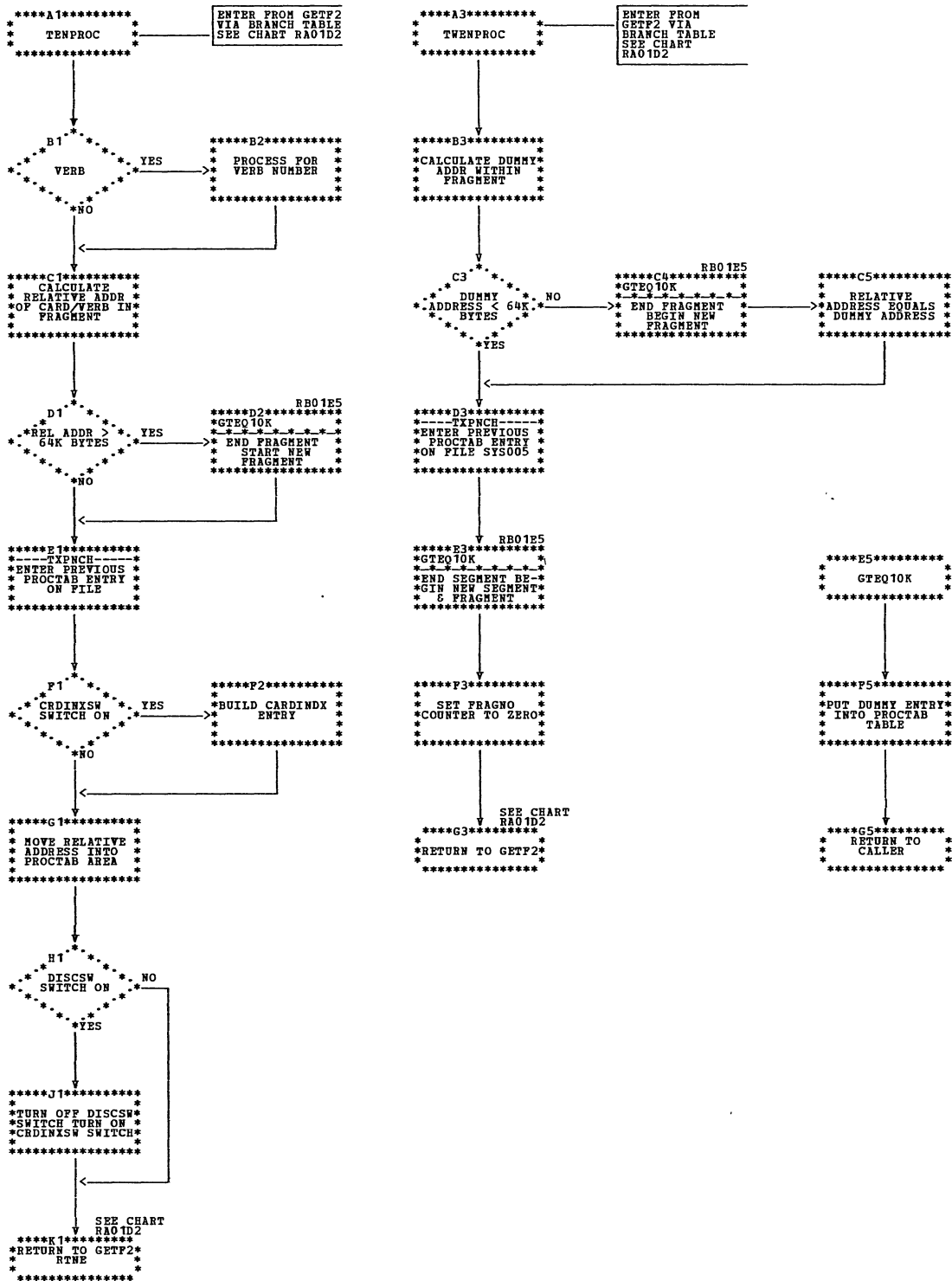


Chart SA. Phase 61 (ILACBL61) Overall Logic (Part 1 of 3)

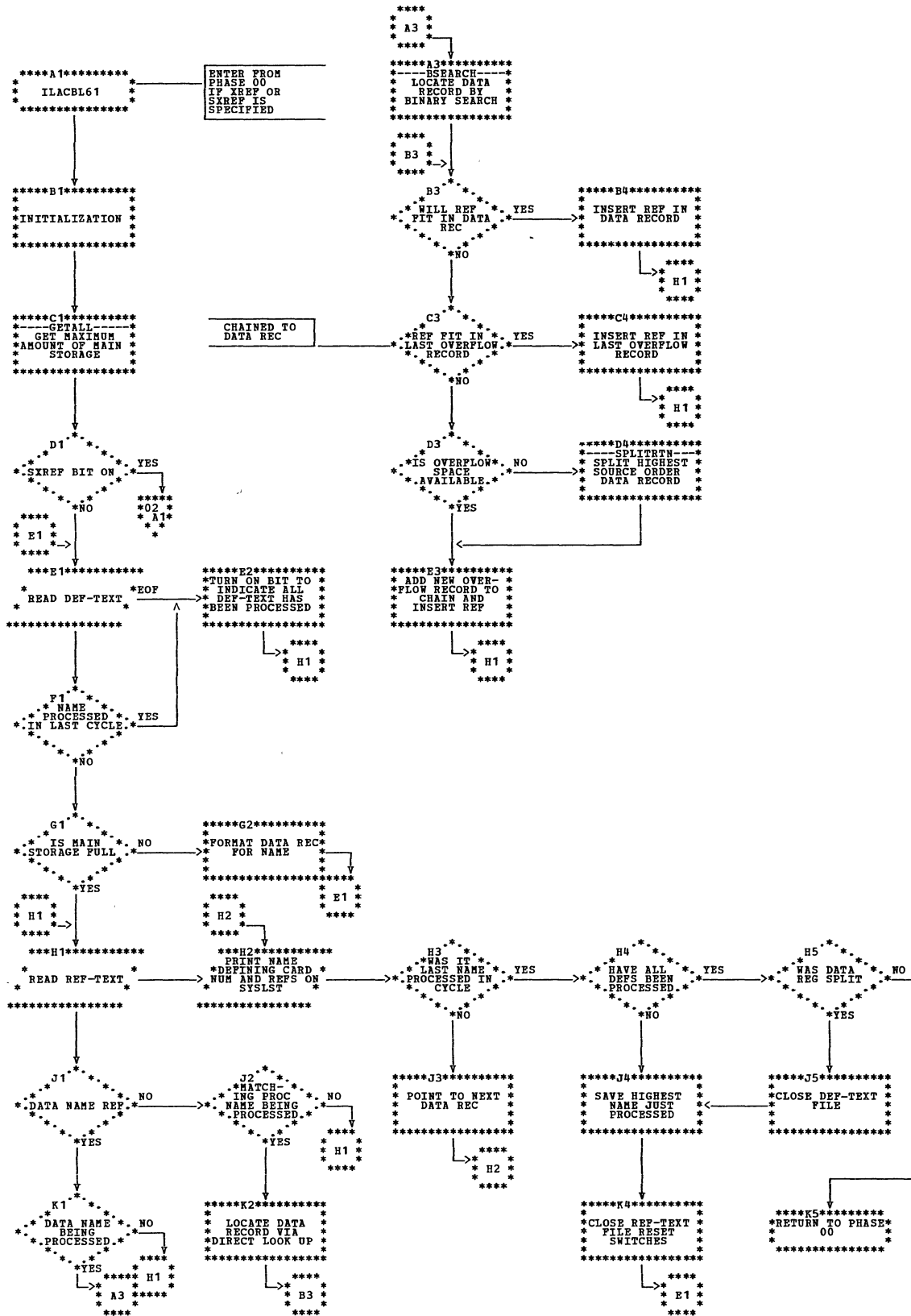




Chart SA.Phase 61 (ILACBL61) Overall Logic (Part 2 of 3)

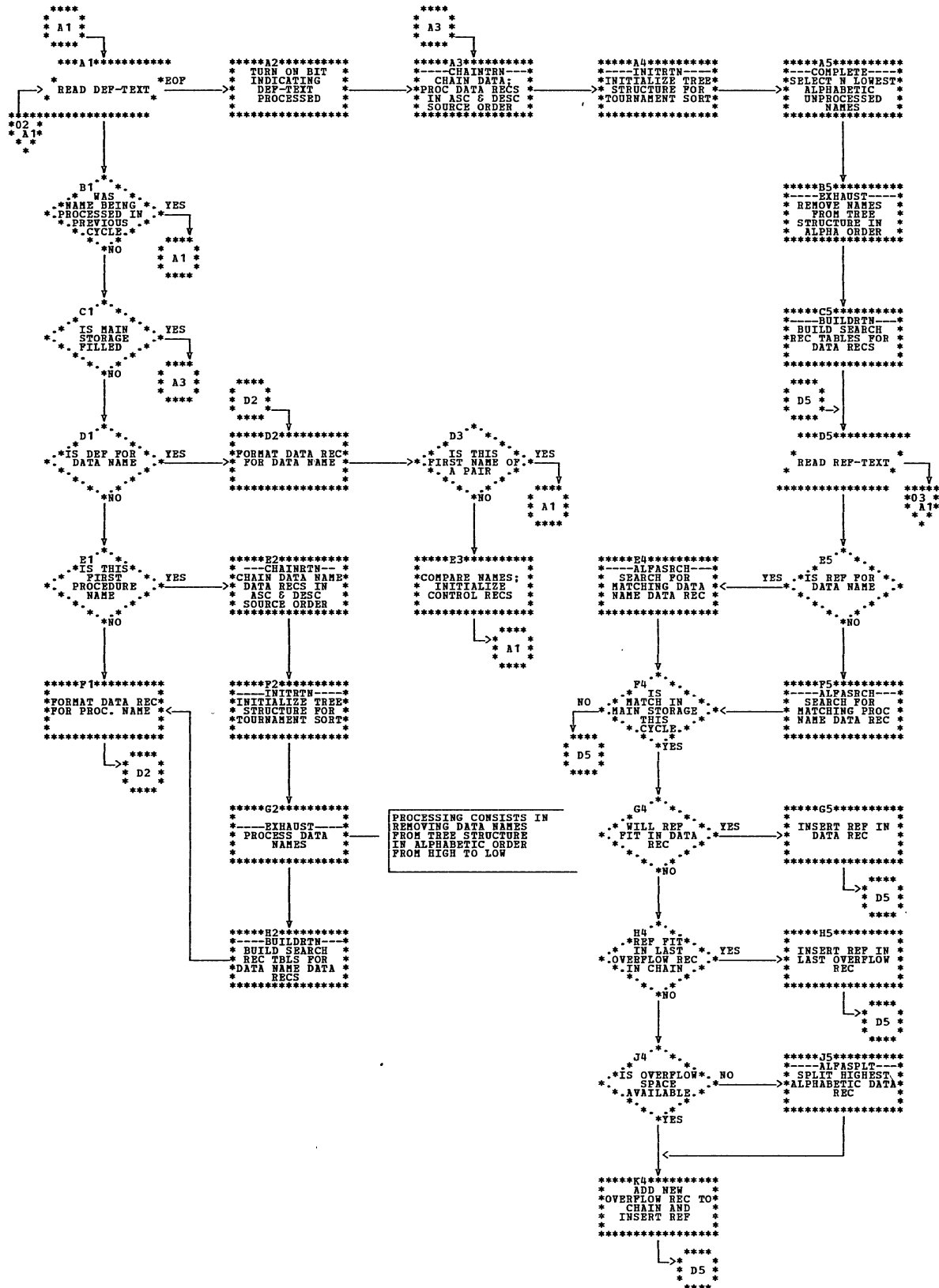


Chart SA. Phase 61 (ILACBL61) Overall Logic (Part 3 of 3)

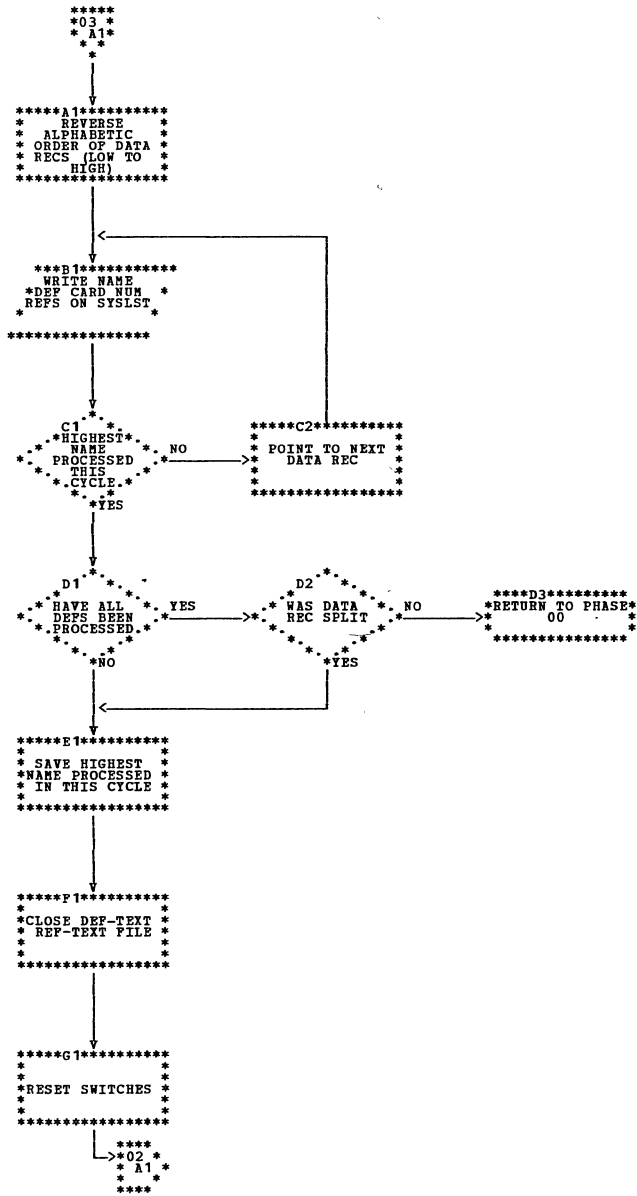


Chart TA. Phase 70 (ILACBL70): Overall Logic

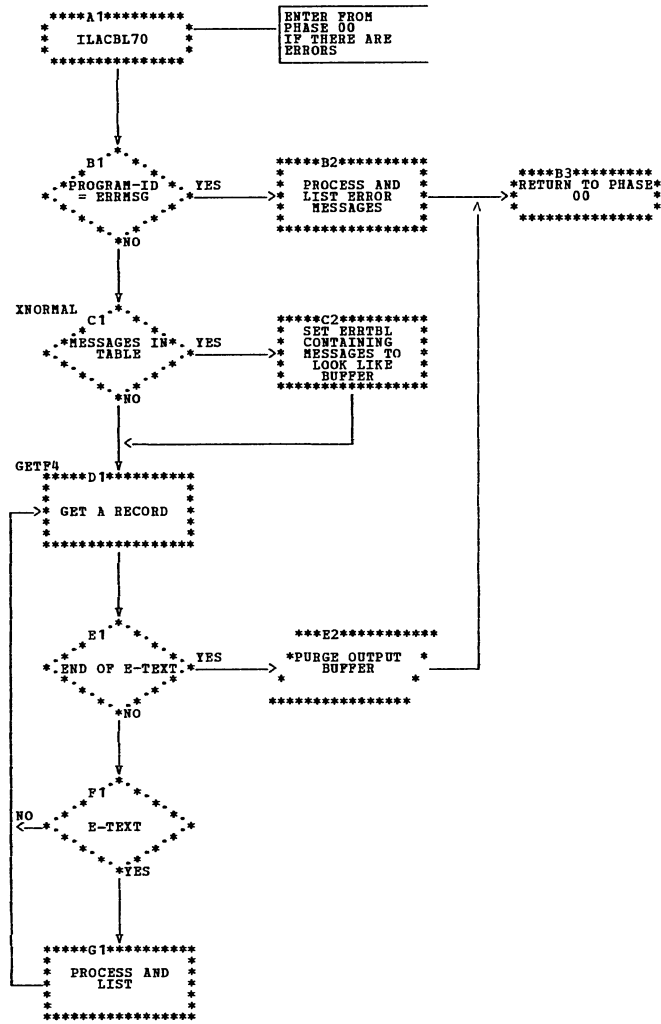


Chart UA. DET-ROUT Subroutine, Report Writer Subprogram

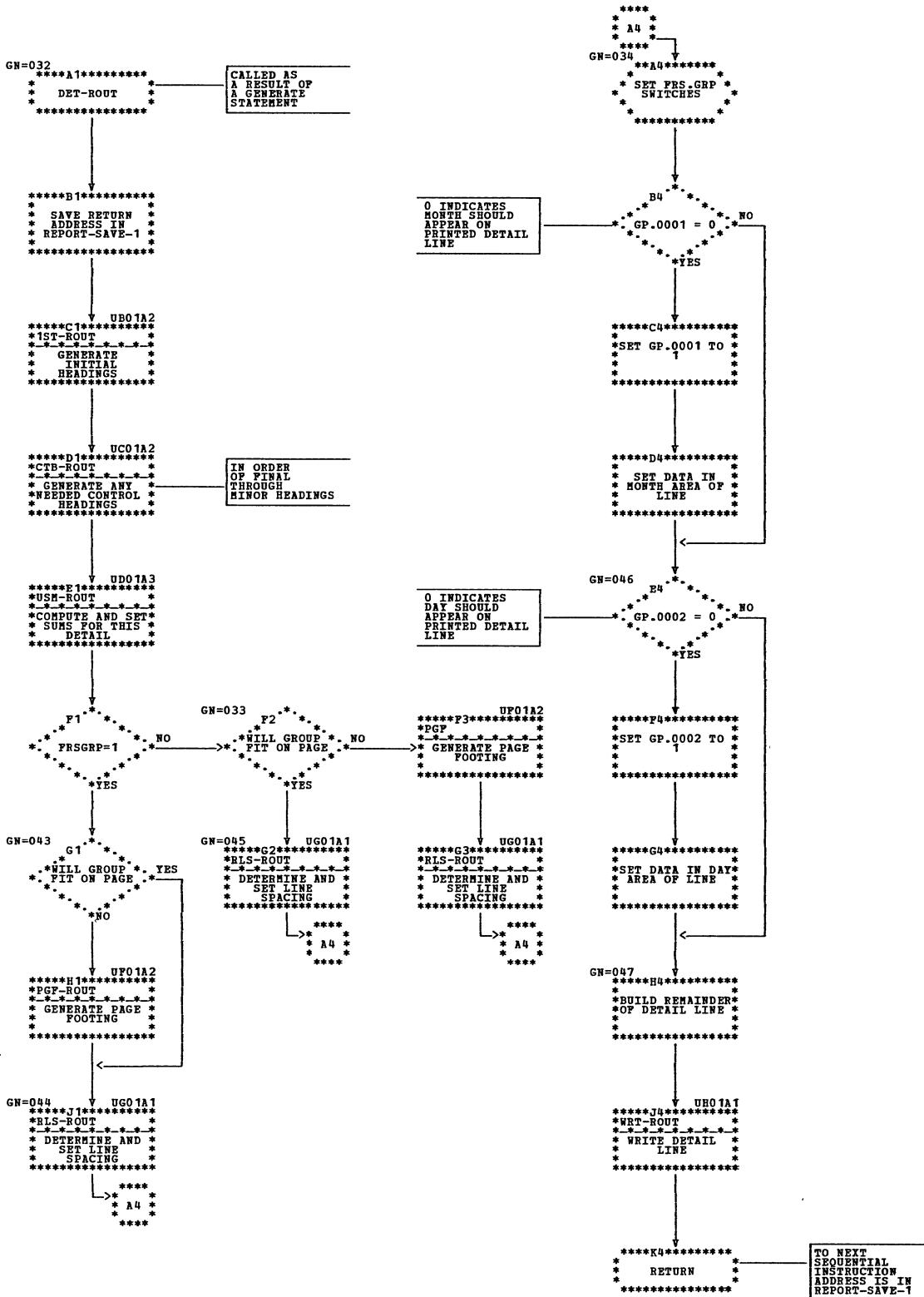


Chart UB. 1ST-ROUT Subroutine, Report Writer Subprogram

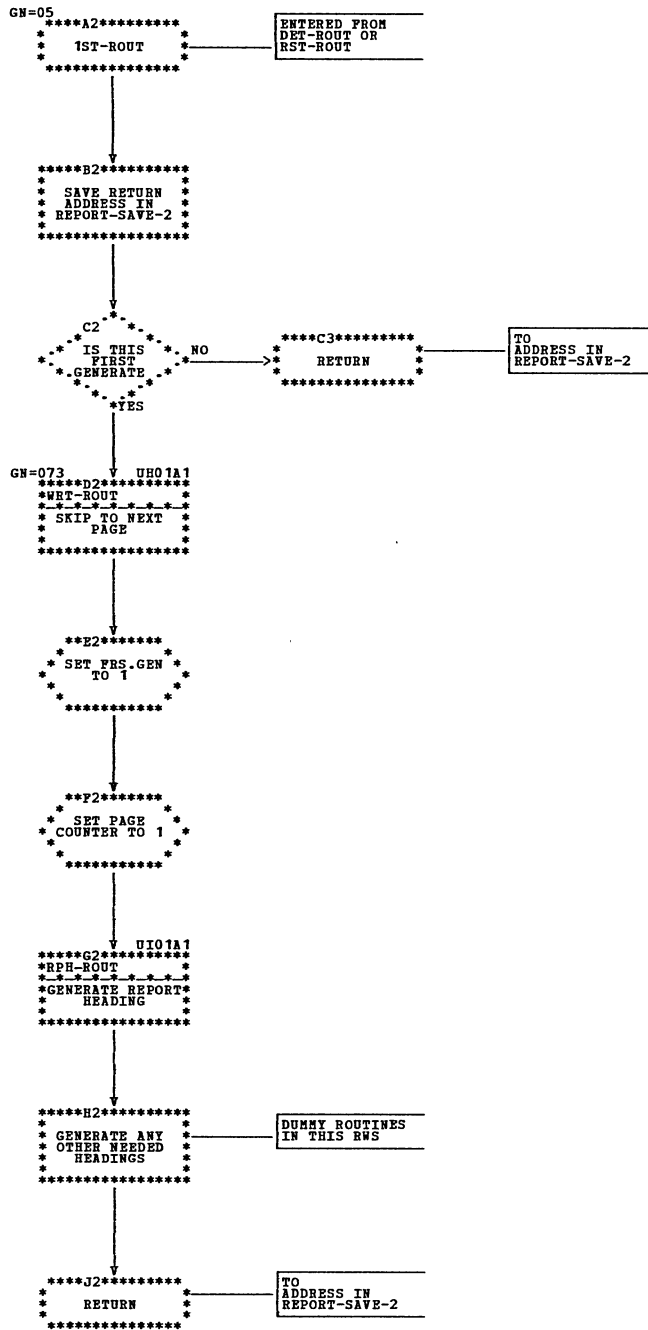


Chart UC. CTB-ROU1 Subroutine, Report Writer Subprogram

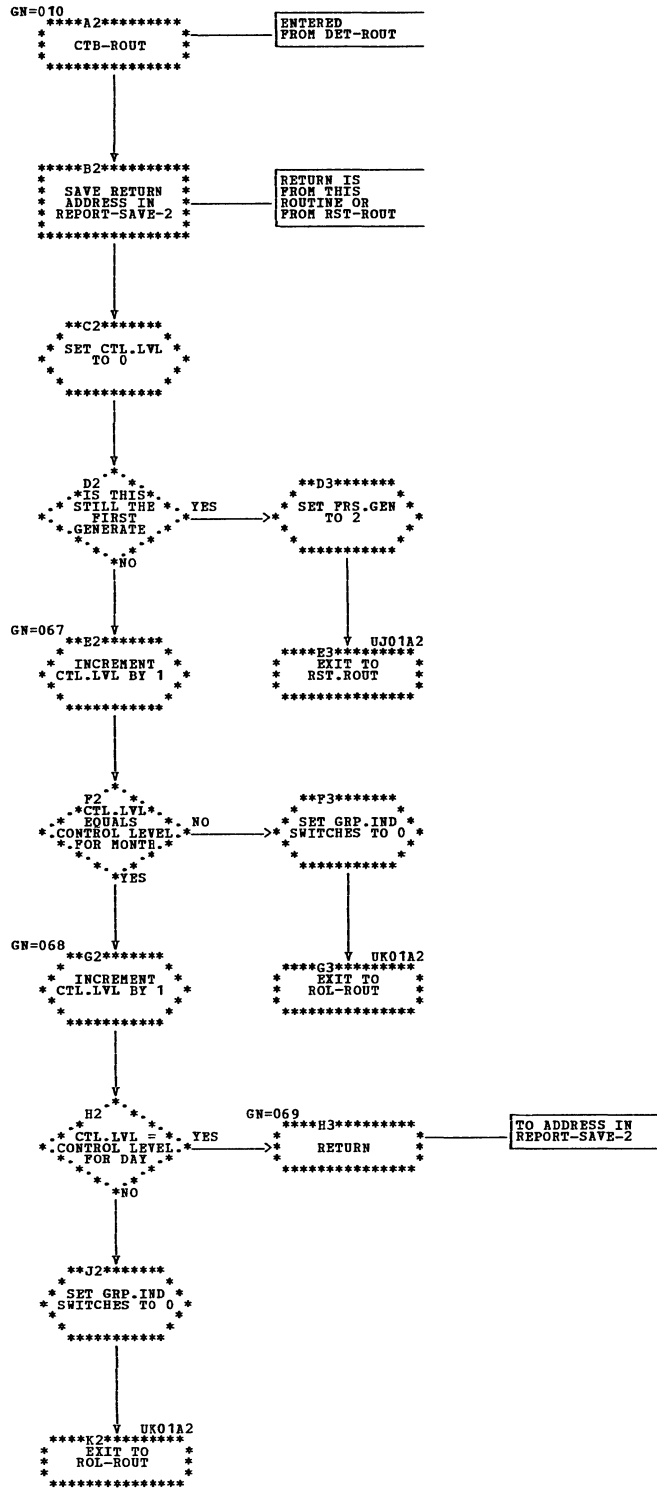


Chart UD. USM-ROUT Subroutine, Report Writer Subprogram

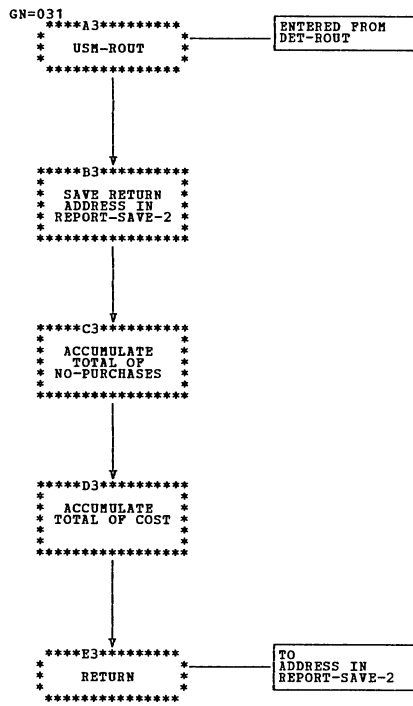


Chart UF. PGF-ROUT Subroutine, Report Writer Subprogram

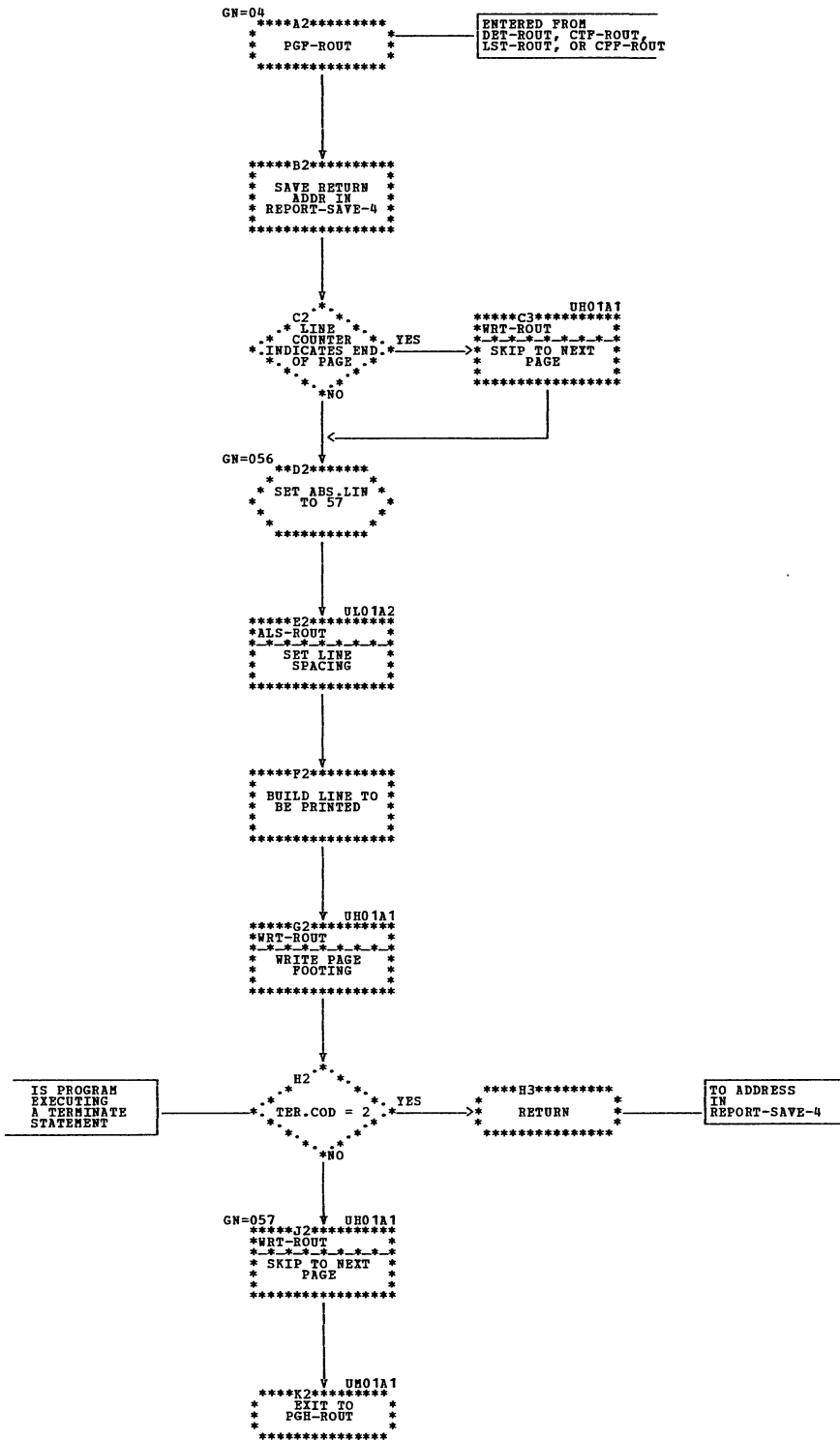




Chart UG. RLS-ROUT Subroutine, Report Writer Subprogram

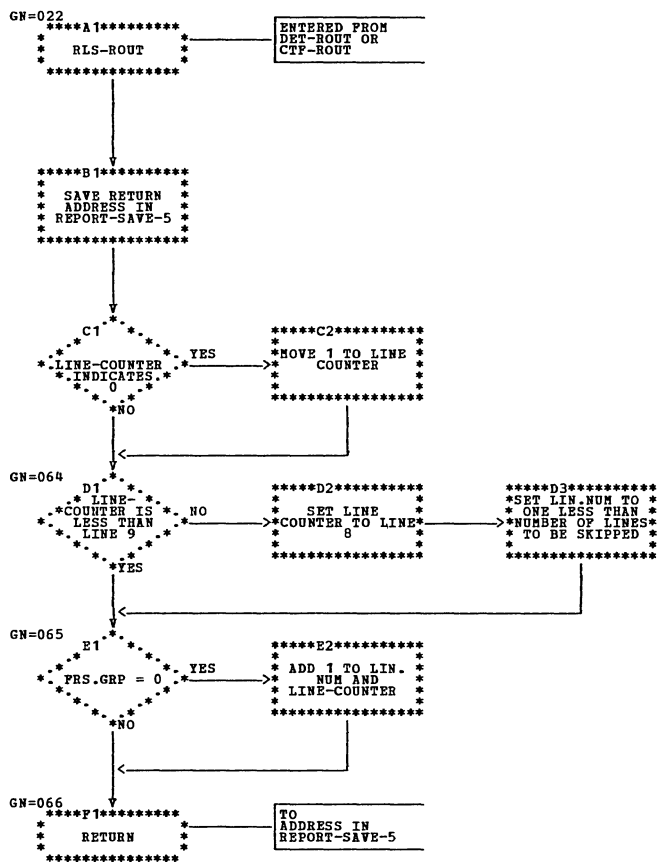


Chart UH. WRT-ROUT Subroutine, Report Writer Subprogram

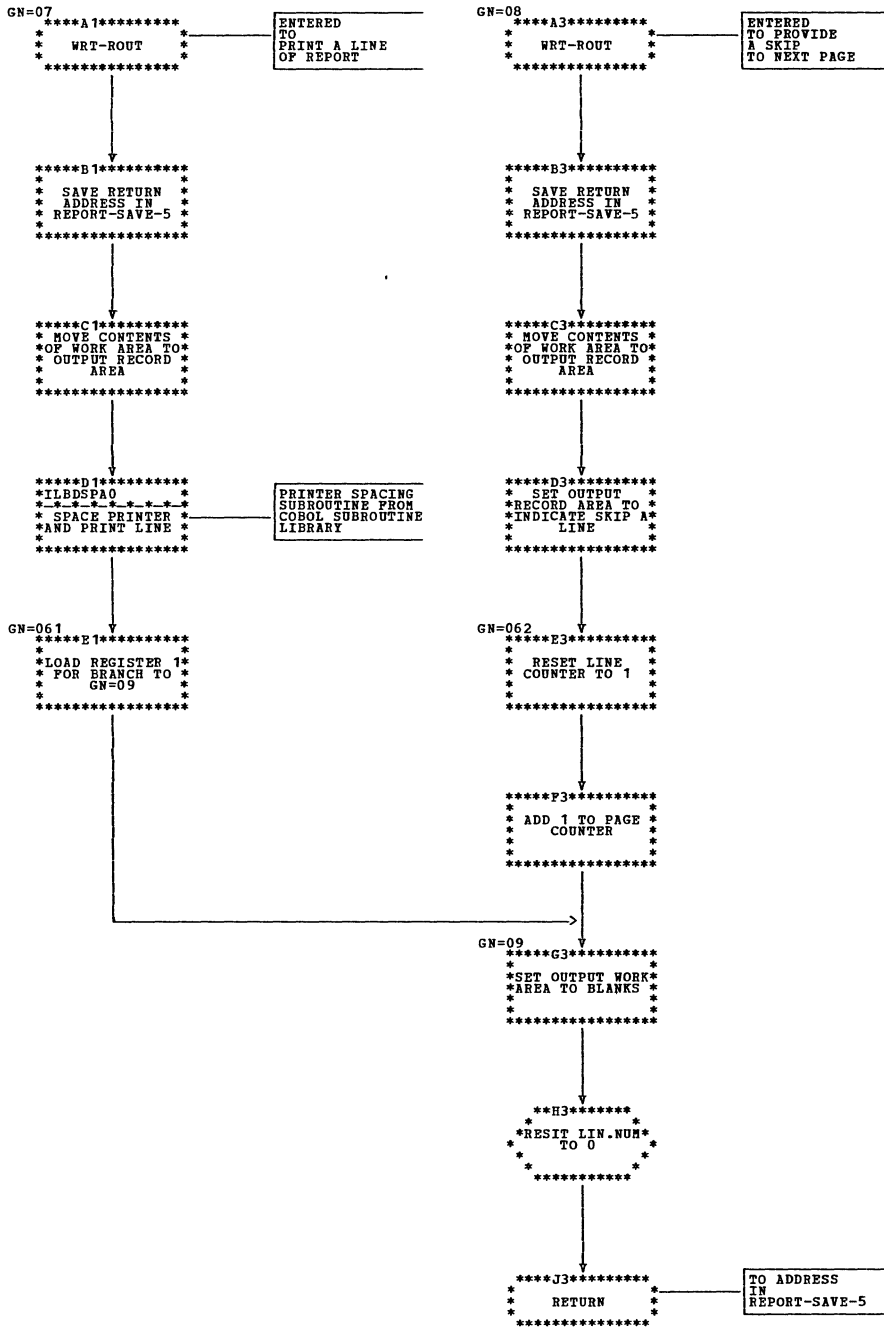


Chart UI. RPH-ROUT Subroutine, Report Writer Subprogram

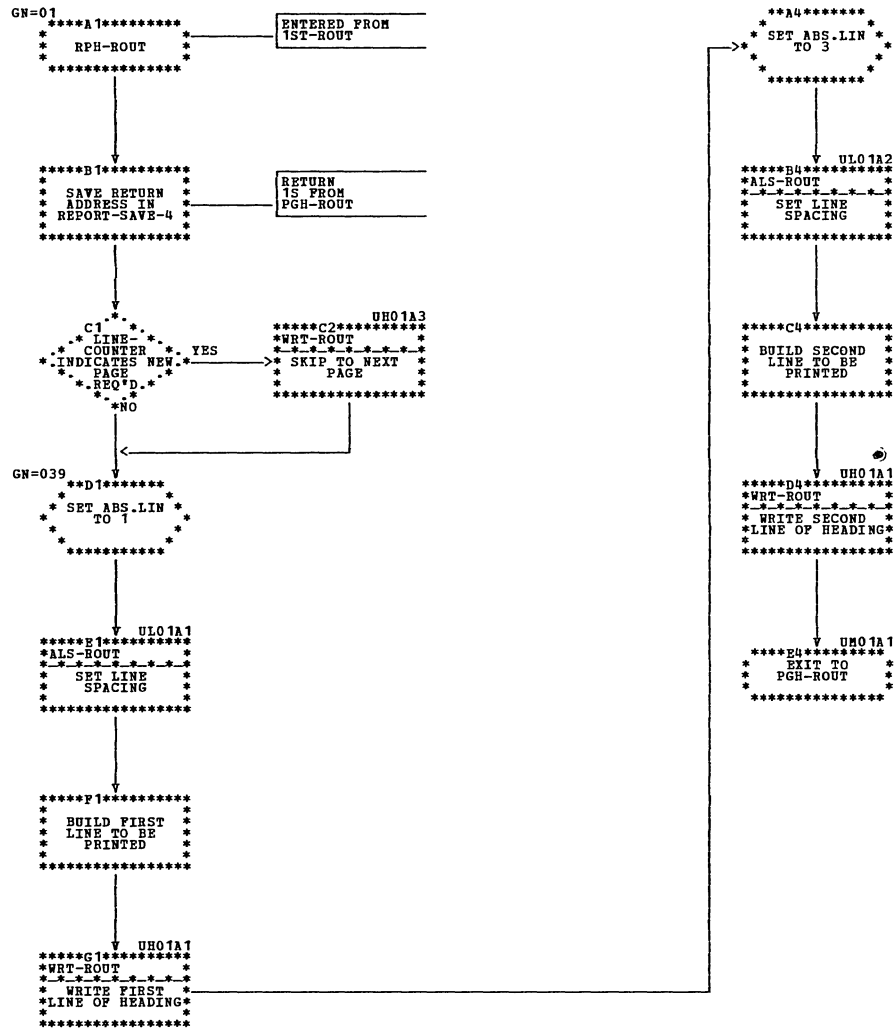


Chart UJ. RST-ROUT Subroutine, Report Writer Subprogram

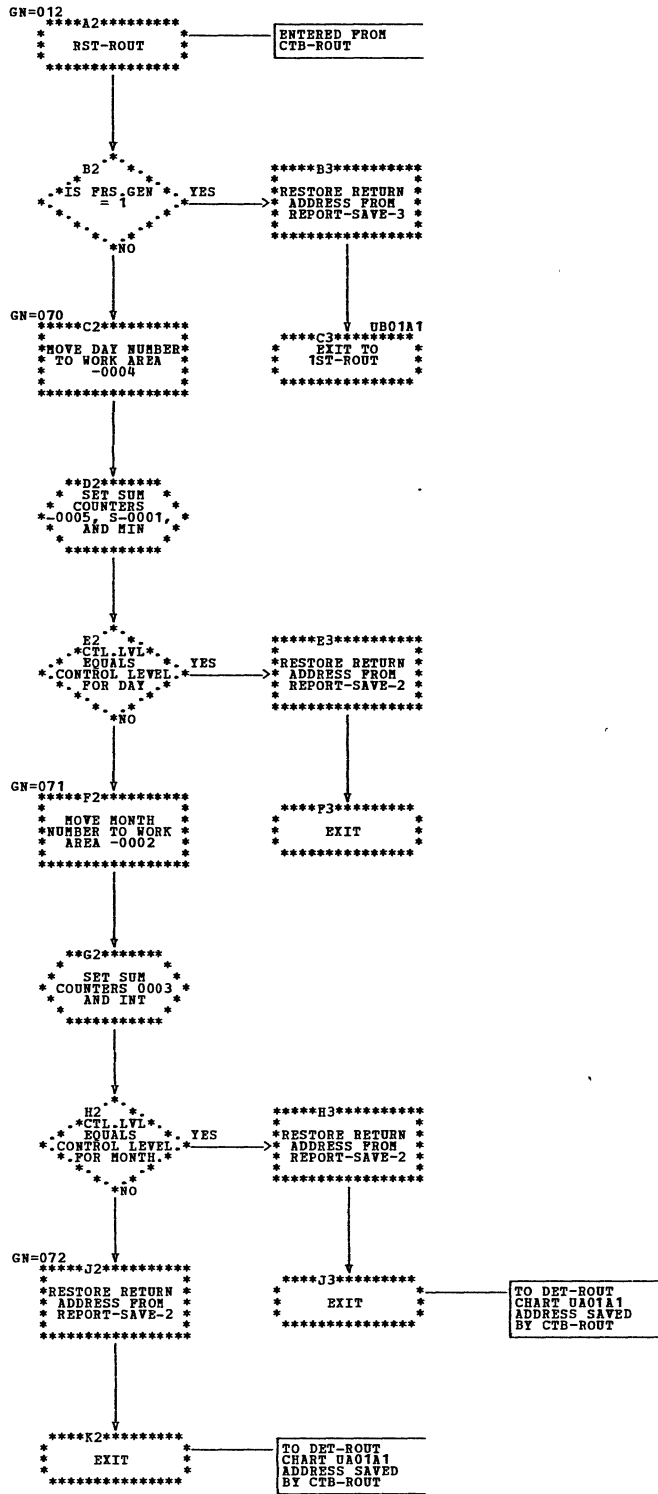


Chart UK. ROL-ROUT Subroutine, Report Writer Subprogram

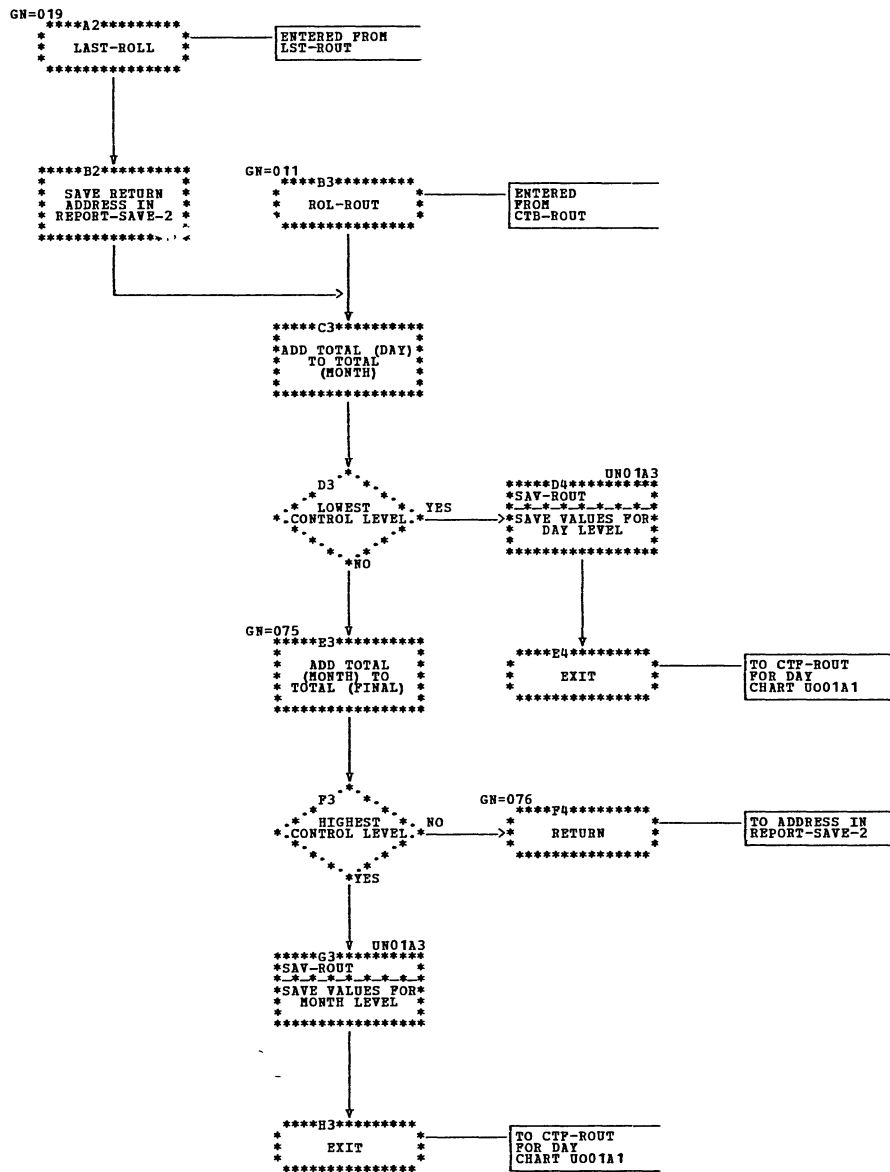


Chart UL. ALS-ROUT Subroutine, Report Writer Subprogram

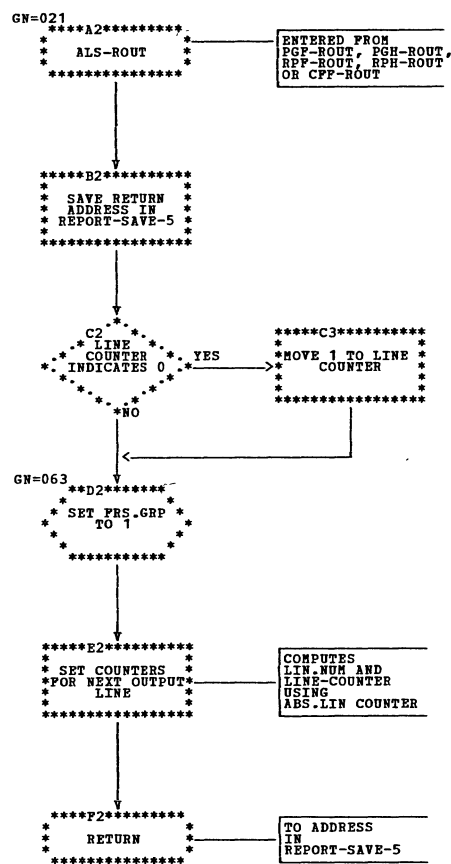


Chart UM. PGH-ROUT Subroutine, Report Writer Subprogram

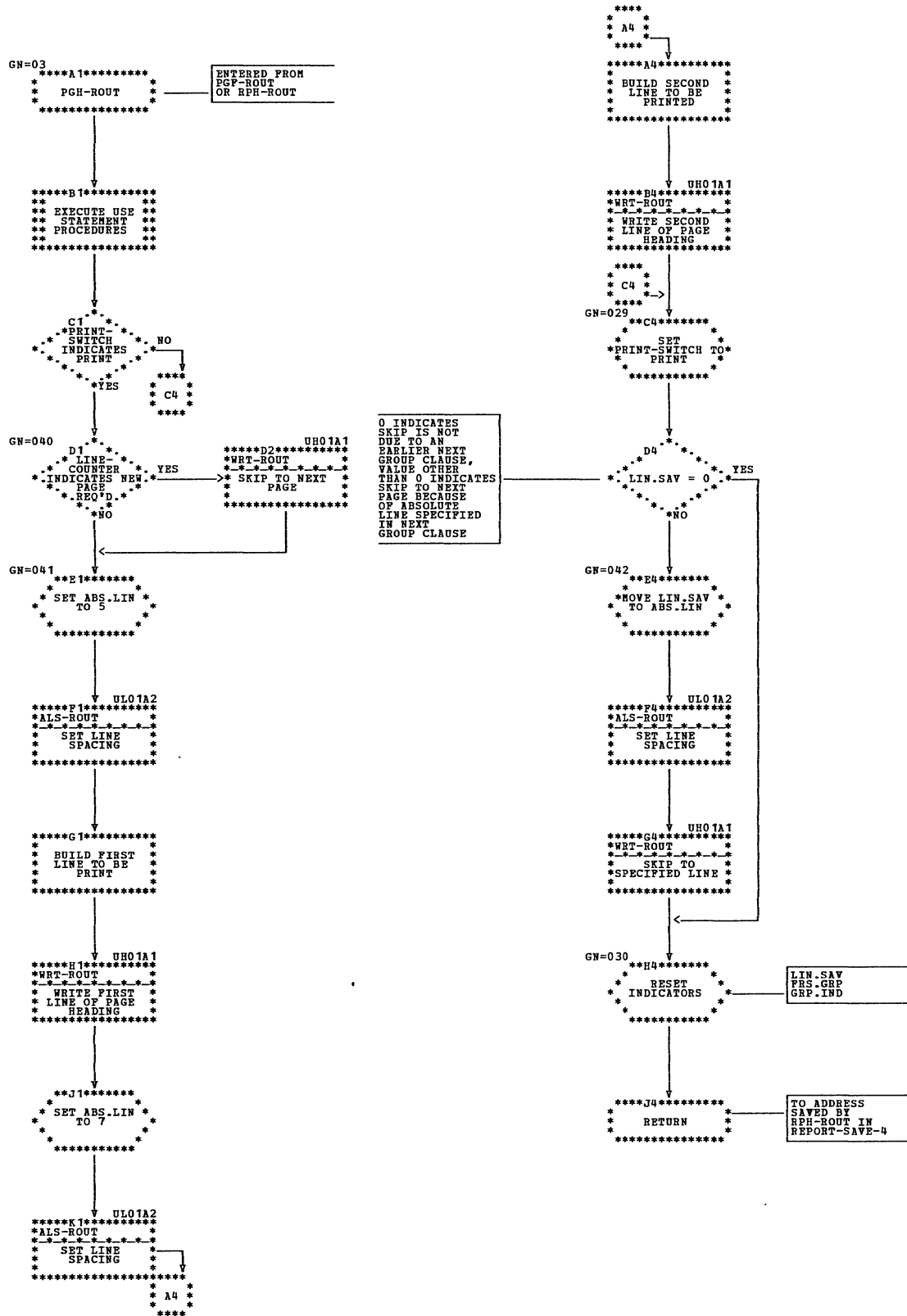


Chart UN. SAV-ROUT Subroutine, Report Writer Subprogram

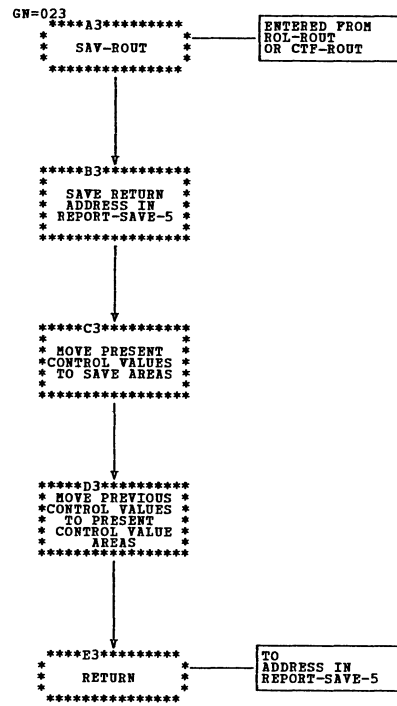




Chart UO. CTF-ROUT Subroutine, Report Writer Subprogram

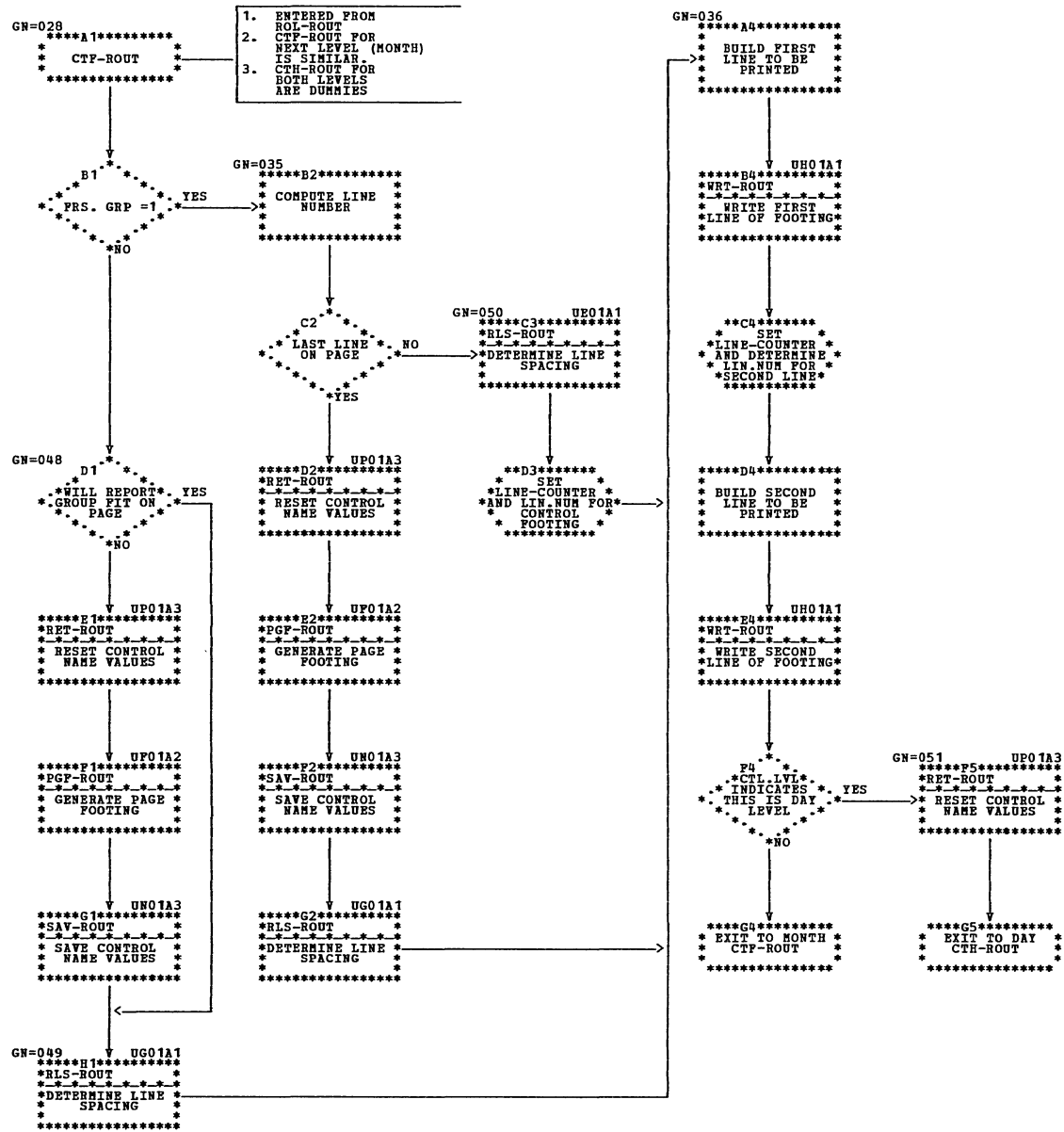


Chart UP. RET-ROUT Subroutine, Report Writer Subprogram

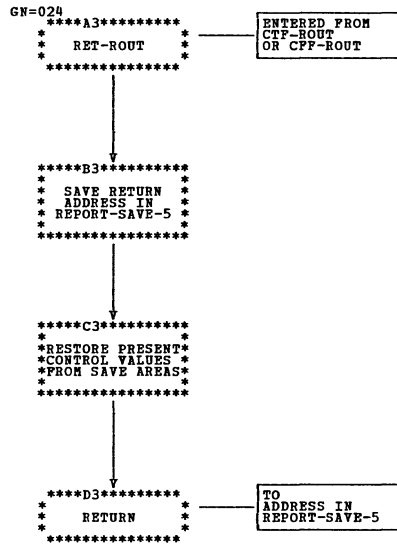


Chart UQ. INT-ROUT Subroutine, Report Writer Subprogram

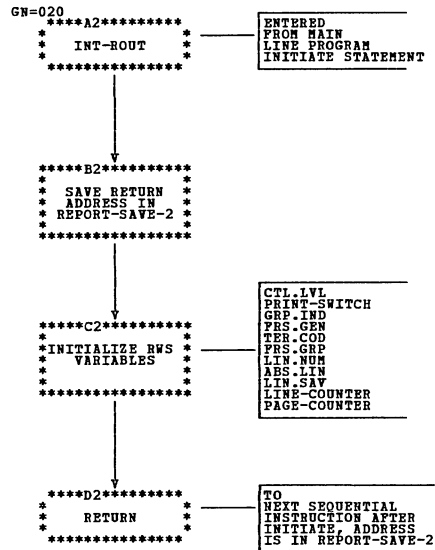


Chart UR. LST-ROUT Subroutine, Report Writer Subprogram

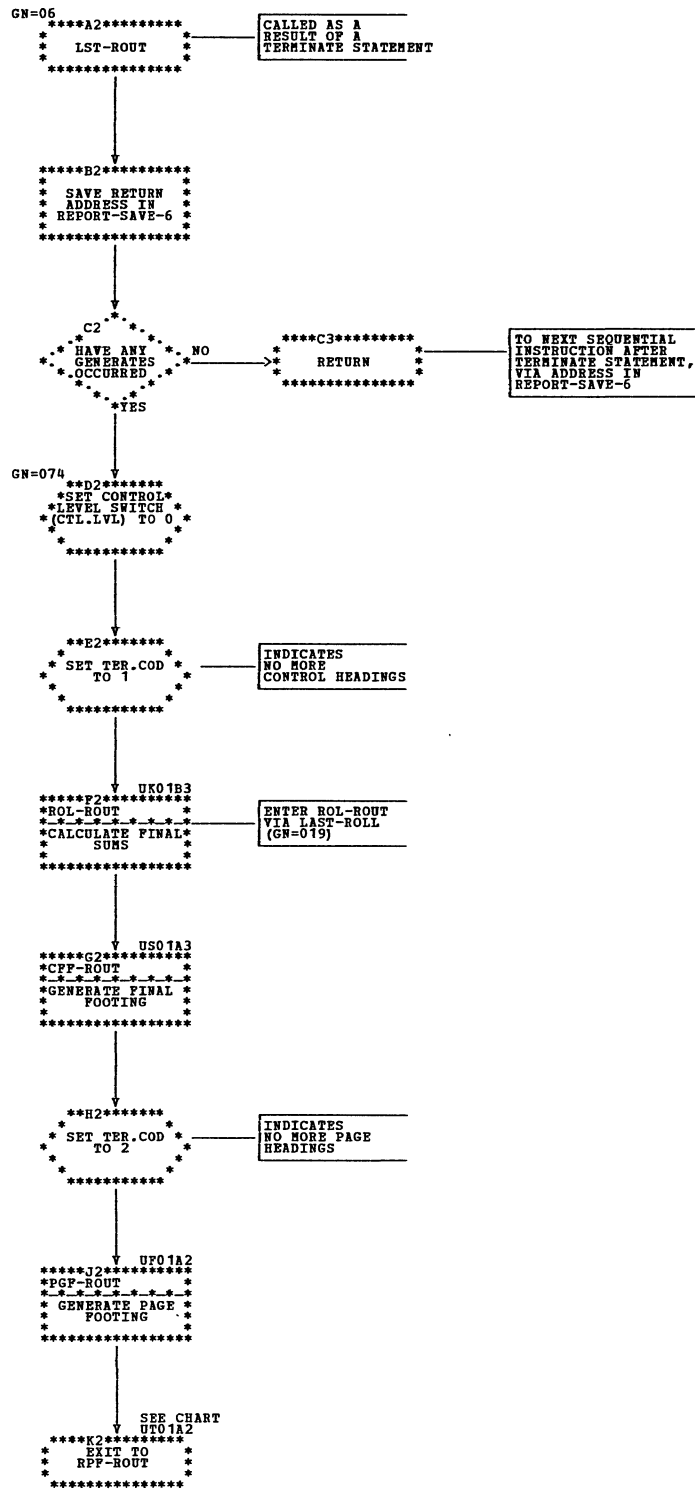


Chart US. CFF-ROUT Subroutine, Report Writer Subprogram

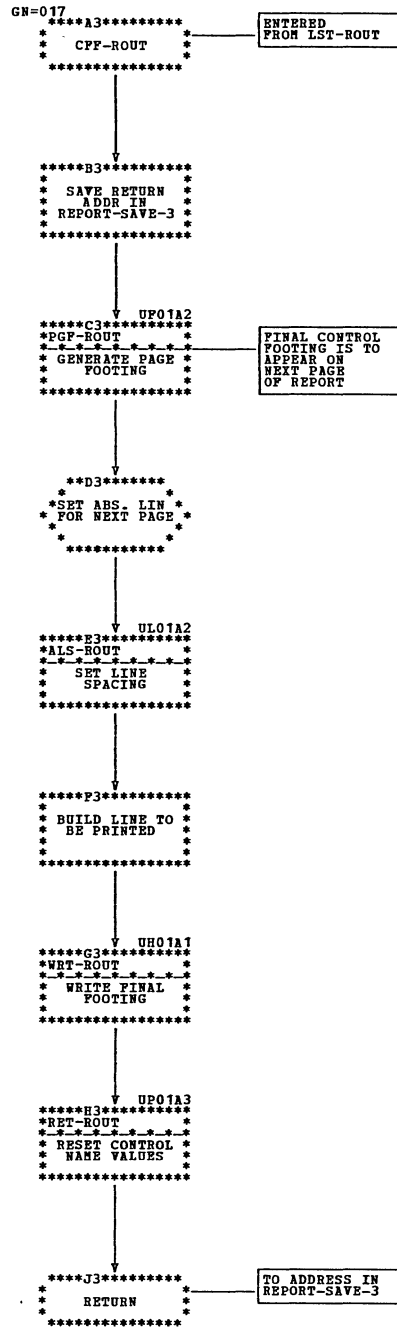
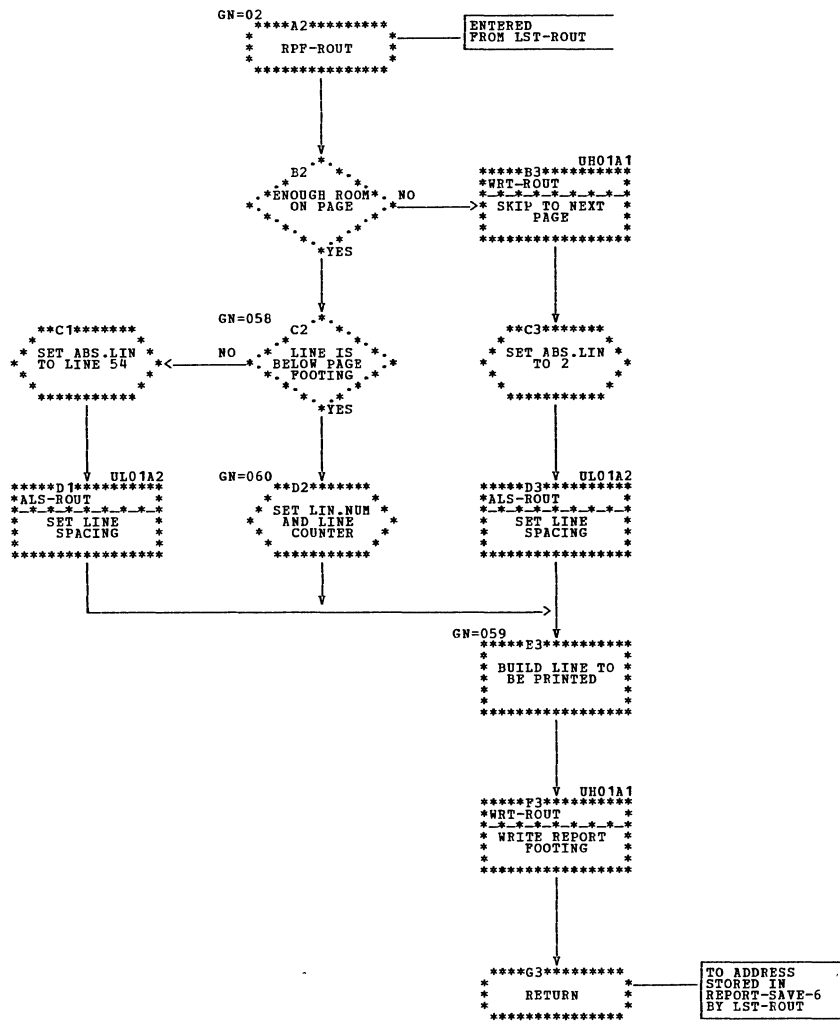


Chart UT. RPF-ROUT Subroutine, Report Writer Subprogram



SECTION 4. DIRECTORY

FLOWCHART LABEL DIRECTORY

<u>Label</u>	<u>Chart</u>	<u>Page</u>	<u>Block</u>	<u>Label</u>	<u>Chart</u>	<u>Page</u>	<u>Block</u>
ADETER	ML	01	B1	DONGP	EB	01	B1
ADINCR	PE	01	F2	DUMTST	CA	01	B1
ADREF	PE	01	A2	D6PN10	NE	01	D5
ADREF	QB	01	A1	D6SR10	NE	01	D4
ASETCPY2	BA	02	E4	D6000	NE	01	E3
BCONDE	HE	01	H5	EINRG	NE	02	E2
BCONDE	HB	01	F4	ELIPR	HG	01	H2
BCONDE	HD	01	F4	END	NC	01	E1
BEGIN	FA	01	B2	END	OC	01	E1
BELEMI	FD	01	A4	ENDIN	AA	07	C3
BLDOB02	IB	01	C3	ENDPTX	ND	01	C3
BLDOB06	IB	01	G3	ENDPTX	OD	01	B2
BLDOB07	IB	01	H3	ENDP13	IB	01	C5
BGROUP	FD	01	J2	ENTNAM	HF	01	G4
BMSRN	FD	01	C1	ENVSCN	CA	01	B3
BNORML	FD	01	D4	EOF	JA	01	E3
BREAD	FD	01	E1	EOF	KA	01	D4
BPASS12	IC	01	F1	EOF	ND	01	C2
BPASS2	IC	01	G1	EOF	PA	01	C3
BSUBRN	FB	01	D3	EOF2	RA	01	D3
BUSAGE	FD	01	G1	EPFT	EB	01	C2
CBASIS	AA	05	A3	EPHD	EB	01	D2
CBENDIN	AA	05	E4	EQUATE	MJ	01	A1
CBEXIT	AA	05	F3	ERROR	JC	01	J2
CBNOFND	AA	05	C5	ERROR	KB	01	G3
CHKDCL	DA	01	C1	ERROR	KB	01	E4
CHKENT	JB	01	G3	EXIT	AA	07	A3
CKCOMP	KB	01	D4	EXITPGM	MK	01	G4
CKPFCP	KC	01	J3	EXITQ	AA	07	B3
CLANAB	MP	01	A1	EXITR	AA	07	B5
CLOSE	NA	01	G3	EXITL	AA	07	B4
CLOSE	OA	01	G3	EXIT1A	AA	07	C5
CLOSER	AA	07	A1	FDSCN	CD	01	E3
CLOSETA	AA	04	A4	FDTEXT	FB	01	C2
CLOSETF	AA	04	C4	FILED	IC	01	E3
COPYIN	JC	01	E2	FLEX1	EB	01	C1
COPYIN	JC	01	D3	FNDL	AA	05	B4
COPYIN	JC	01	F4	FORMLA	KB	01	D1
COPYRN	FD	01	F1	FOURTY8	PA	01	E3
CORRTN	JC	01	G4	FSECT	HB	01	A2
CPPRO	ND	01	D2	FSEND	HG	01	C3
C0	PA	01	F3	FSEQ	NE	02	C2
ClREF	PF	01	A1	FSTXT	HB	01	C3
DCLSCN	DA	01	C2	FST000	HG	01	C4
DDSCN	CA	01	E3	FST000	HG	01	F3
DEBUG	ML	01	A1	FTER	NE	02	H2
DELIM	DA	01	D1	GCKOP3	LC	01	D1
DICENT	DA	01	D3	GCKOP3	MR	01	D1
DICSCN	JB	01	B2	GC3	BA	02	H1
DICTBD	HG	01	B2	GDDAGN	LC	01	B3
DICTENTR	ID	01	C1	GDOAGN	MR	01	B3
DICTP1	IC	01	H1	GENA	DA	01	G3
DIR	HA	01	D1	GENDAT	JC	01	J3
DIR010	HA	01	E1	GENRATOR	ME	01	D4
DISPLAY	MJ	01	A1	GENOP	JC	01	H3

Licensed Material - Property of IBM

<u>Label</u>	<u>Chart</u>	<u>Page</u>	<u>Block</u>	<u>Label</u>	<u>Chart</u>	<u>Page</u>	<u>Block</u>
GENSTR	KC	01	G4	INTERLUD	AA	06	E5
GET	ND	01	B1	ISPRNC	KB	01	C1
GET	ND	01	D3	ITEMRN	FD	01	H1
GET	OD	01	B1	JCANCEL	AA	02	F5
GET	PA	01	C2	KILSUB	LD	01	A4
GETCARD4	BA	02	F1	LDSCN	CA	01	G4
GETCOPY	BA	02	E2	LDTXT	HE	01	A2
GETCOPY4	BA	02	E4	LINKA	AA	06	B3
GETCRD	EA	01	C2	LINKB	AA	06	A3
GETDLM	CA	01	E1	LINKR	AA	06	B4
GETF2	RA	01	D1	LINKST	FC	01	A3
GETF4	TA	01	D1	LOADLIT	MG	01	A4
GETN	IB	01	B1	LOGNWT	IC	01	B1
GETNXT	HF	01	C1	LSECT	HC	01	A3
GETNXT	JC	01	E3	MACPRO	ND	01	H2
GETNXT	JC	01	B1	MACRO	PA	01	D3
GETOUT	FD	01	E4	MOVEMOD	BA	01	D4
GETPTR	HE	01	G4	MOVE4	MF	01	B1
GETPTR	HD	01	D4	MPUT	BA	01	B5
GETPTR	HB	01	D4	NOGET	IC	01	E1
GETPVN	KC	01	D2	NOTBASIS	BA	02	E1
GETSL	AA	05	D3	NTSL	AA	05	C2
GIDENL	ND	01	K2	NXTFILE	BA	01	F3
GINIT2	NE	02	B2	OD2FND	IB	01	D1
GINIT3	NE	02	D2	OPPRO	OD	01	C2
GLOSRY	JB	01	A1	PDATEX	NE	01	A1
GNDDEF	PC	01	A2	PDT020	NE	01	C2
GNREF	PF	01	G1	PDT030	NE	01	D3
GNOPT	LC	01	C3	PERFORM	KC	01	A1
GO	MN	01	A1	PERFORM	MI	01	A1
GO DEPENDING	MO	01	A3	PGTINT	OB	01	G3
GOBACK	ML	01	F3	PGTINT	NB	01	G3
GODEPL	MO	01	A1	PHASEND	IA	01	F2
GODEPM	MN	01	H2	PHCTRL	JC	01	A1
GOSYSGO	AA	01	B1	PHINIT	JA	01	B3
GRIPR	HG	01	G3	PHINIT	EA	01	B2
GSPICT	FD	01	B2	PHINIT	KA	01	B3
GTEQ10K	RB	01	E5	PHTERM	FA	01	G2
IDDSCN	CA	01	C1	PH5BVB	LA	01	G2
IDENT	KA	01	C3	PH5CTL	LA	01	B2
IDLAN	KA	01	F3	PH5CTL	MA	01	B2
IF	MM	01	A1	PH65	RA	01	B2
IFANAL	MP	01	A4	PICTAN	FD	01	B5
IFERR	KB	01	E1	PLUS1	LC	01	A3
IFSO	KB	01	F1	PLUS1	MR	01	A3
IHNAM	DA	01	D2	PNBRR0	ND	01	F2
IMGEN	MK	01	G1	PNDEF	PD	01	A2
IMINIT	MK	01	A1	PNREF	PF	01	E3
INIT	HA	01	A1	PNUPRO	NB	01	H3
INIT1	NE	02	G2	POINT0	ND	01	J3
INSERT	NB	01	J3	PREPROC	BA	02	A1
INSERT	NC	01	F2	PRFTWO	NC	01	A1
INSERT	NC	01	A5	PRFTWO	OC	01	A1
INSERT	NC	01	G2	PRINT	JB	01	D3
INSERT	NC	01	H2	PRINT	JB	01	E3
INSERT	NC	01	J2	PROC01	EA	01	E2



<u>Label</u>	<u>Chart</u>	<u>Page</u>	<u>Block</u>	<u>Label</u>	<u>Chart</u>	<u>Page</u>	<u>Block</u>
PROC02	EA	01	G2	SKPLNK	AA	02	E5
PROC77	HG	01	E3	SRCHTB	FD	01	D2
PTSL	AA	05	B3	STARTPP	BA	02	C1
PUNCH	NC	01	F3	START0	BA	02	D2
PUNCH	NE	02	E3	STATIC	NB	01	K4
PUNCH	OC	01	F3	STRSCH	JC	01	C4
PURGE	AA	06	C4	TENPROC	RB	01	A1
PUTDEF	MD	01	A1	TERM	HA	01	E2
PUTEQU	MD	01	F4	TESTSB2	ID	01	D2
QIFOUND	IB	01	E1	TESTSB3	ID	01	F2
QUAL	HD	01	C4	TESTSB4	ID	01	H2
QUAL	HE	01	H3	TESTSB6	ID	01	D3
RC4	QB	01	F1	TESTSB8	ID	01	J3
RC8C	QB	01	A3	THRESUBS	ID	01	C4
RDSCAN	EA	01	D2	TGTINT	OB	01	F3
RDSYN	HG	01	D4	TGTINT	NB	01	F3
RD001	QB	01	F3	TRACE	ML	01	A3
READ	AA	02	A1	TRANSFORM	MI	01	A4
READFN	JC	01	C3	TRMNATE	AA	02	H3
READF2	NC	01	C1	TWENPROC	RB	01	A3
READF2	OC	01	C1	TWOSUBS	ID	01	C3
READF4	HF	01	B1	TXPNH	NC	01	G3
READLIB	AA	05	A1	TXPNH	OC	01	G3
READOPTS	BA	01	G1	UNTIL	KC	01	G3
READQ	AA	02	A5	UNTIL	KC	01	H4
READY	ML	01	H1	UPSI	HA	01	C2
REDEF	HG	01	D3	VALGEN	FD	01	J4
RELEASE	NB	01	K3	VALGEN	FD	01	C5
RELEASE	NC	01	H3	VERB	JC	01	D2
RELEASE	NE	01	F5	VNDEFR	ND	01	E2
RELEASE	OC	01	H3	VRBSCN	DA	01	E3
RENAMS	HF	01	F4	WLVSCN	CA	01	F4
RENML0	IC	01	J4	WOUT	AA	04	C1
REPORT	FC	01	A5	WRITE	AA	04	B1
REPORTD	IC	01	G5	WRITEA	AA	04	A1
RESET	MK	01	A4	WRITE5	IC	01	B2
REWIND	NC	01	D3	WSDCT	HC	01	A1
REWIND	ND	01	C5	WSTSCT	FC	01	A1
REWIND	NE	01	B3	XITXIT	EB	01	C4
RSECT	HD	01	A2	XIT1	EB	01	F1
SCATREN	IA	01	C3	XIT2	EB	01	G1
SDTEXT	FB	01	D2	XIT2A	EB	01	G2
SDTXT	HB	01	D3	XIT3	EB	01	H1
SEARCH	JC	01	G2	XIT4	EB	01	J1
SEGCAL	MQ	01	A3	XIT5	EB	01	A4
SEGCAL3	MQ	01	F2	XIT6	EB	01	B5
SEGN0TE	AA	06	D1	XIT7	EB	01	A5
SEGN0TE0	AA	06	F2	XNORMAL	TA	01	C1
SEGN0TE2	AA	06	G1	XREF	HB	01	E4
SEGPNT	AA	06	A1	XREF	HD	01	E4
SEGPCOC	ND	01	J2	XREF	HE	01	G5
SETBUF6	BA	01	A3	XREF	NC	01	D2
SETLEN	MG	01	A2	XSPRO	LD	01	A1
SE6000	ND	01	A1				
SE6000	OD	01	A1				
SE6025	ND	01	D1				



TABLES USED BY PHASES

Table and TIB Number		
Phase	Built or Changed by Phase	Referenced Only
01	REPTAB (29)	
10	CKPTBL (8), ENVTBL (3), FNTBL (10), INDXTB (27), KEYTAB (26), OD2TBL (9), PIOTBL (7), P1BTBL (2), QLTABL (1), QNM TBL (2), RCDTBL (11), RWRTBL (13), SATBL (5), SPNTBL (21), SRATBL (6), SSATBL (7), UPSTBL (25)	
12	CTL TBL (14), DETTBL (17), GCNTBL (21), NPTTBL (18), PIOTBL (7), P1BTBL (2), QALTBL (23), QLTABL (1), RNMTBL (12), ROLTBL (15), ROUTBL (16), SMSTBL (28), SNMTBL (35), SRCTBL (22), SUMTBL (19)	FNTBL (10), RWRTBL (13), SPNTBL (21)
11	DICOT (20), GVFNTBL (4), GVNMTBL (3), PIOTBL (7), PNQTBL (6), PNTABL (5), QLTABL (1), RCDTEL (11), RNMTBL (12)	DETTBL (17), FNTBL (10), P1BTBL (2), ROUTBL (16), RWRTBL (13), SPNTBL (21)
20	VALGRP (6), VALTRU (33), LAP TBL (13)	
22	DICOT (20), FDTAB (28), GPLSTK (10), INDKEY (31), MASTODO (13), OBJSUB (5), OCCTBL (2), QFILE (23), QITBL (22), QRTN (21), QVAR (24), QSBL (1), RDFSTK (11), RENAMTB (3), RNMTBL (12), SRCHKY (34), VARLTBL (15)	OD2TBL (9), UPSTBL (25), VALGRP (6), VALTRU (33)
21	ASCTAB (3), BLTABL (27), BUFTAB (29), CKPTBL (8), IND2TBL (35), SDSRATBL (11), SRAMAX (10), SRATBL (9)	FDTAB (28), DICOT (20), PIOTBL (7)
25	OCCTBL (2), ODOTBL (14), VARLTBL (15)	DICOT (20), MASTODO (13), OD2TBL (9), QITBL (22), QRTN (22), RENAMTB (3)
30	QFILE (23), QVAR (24)	DICOT (20), INDKEY (31), VALTRU (33), QSEL (1)
40	DBG TBL (13), DEFSBS (18), KEYTBL (20), PFMTBL (12), PNOUNT (14), PSHTBL (17), PSIGNT (15), PTRFLS (16), SETTBL (21), STRING (9), VARYTB (10), VNTBL (11)	
50	BLUSTBL (10), XAVAL (2), XINTR (1), XSCRPT (3), XSSNT (4)	
51	BLUSTBL (10), GNCALTBL (16), IOPTBL (5), PNUTBL (6), SEG TBL (15)	CKPTBL (8)
60	CONDIS (14), CONTBL (9), CVIRTB (12), ERRTBL (10), FILTBL (2), GNTBL (8), LTLTBL (4), PNTBL (7), QTBL (3), RLDTBL (NONE), TGTADTBL (18), VIRPTR (13), VNPTY (17)	PNUTBL (6), SEG TBL (15)
61	CNTLTBL (none), DATATBL (none), OFLOTBL (none)	

Figure 62. Tables Used by Phases (Part 1 of 2)

Table and TIB Number		
Phase	Built or Changed by Phase	Referenced Only
62	BLASGTBL (16), ELVNTEL (23), CONDIS (14), CONTBL (9), CVIRTB (12), DRPTBL (24), DRPLTBL (25), FILTBL (2), GNATBL (8), GNFWDBTB (21), GNLABTBL (19), LTLTBL (4), PNATBL (7), PNFWDBTB (20), PNLABTBL (18), VIRPTR (13), VNPNTBL (29), VNPTY (17)	BLUSTBL (10), PNUSTBL (6), SEGTBL (15)
63	GNLBDTBL (27), PNLBDTBL (26), QGNTBL (24), RLDTBL (28), VNPTY (17)	BLASGTBL (16), BLVNTBL (23), DRPLTBL (25), GNATBL (8), GNLABTBL (19), PNATBL (7), PNLABTBL (18), SEGTBL (15), VNPNTBL (29)
64	ERRTBL (10), QTBL (3), RLDTBL (28),	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 (18)
70		ERRTBL (10)
80		

Figure 62. Tables Used by Phases (Part 2 of 2)

LINKAGE EDITOR MAP

005 LINKAGE EDITOR DIAGNOSTIC OF INPUT

```

ACTION TAKEN: MAP REL-----
LIST INCLUDE ILACRVS
LIST PHASE FCOBOL,S 0001
LIST INCLUDE ILACBL00----- 0002
LIST INCLUDE ILACBL01 0003
LIST PHASE FCOBOL05,CLIB(FCOBOL) 0004
3 LIST INCLUDE ILACBL05----- 0005
LIST PHASE FCOBOL06,CLIB(FCOBOL) 0006
LIST INCLUDE ILACBL06 0007
4 LIST PHASE FCOBOL07,CLIB(FCOBOL)----- 0008
LIST INCLUDE ILACBL07 0009
LIST PHASE FCOBOL08,CLIB(FCOBOL) 0010
5 LIST INCLUDE ILACBL08----- 0011
LIST PHASE FCOBOL10,CLIB(FCOBOL) 0012
LIST INCLUDE ILACBL10 0013
6 LIST PHASE FCOBOL12,CLIB(FCOBOL)----- 0014
LIST INCLUDE ILACBL12 0015
LIST PHASE FCOBOL11,CLIB(FCOBOL) 0016
7 LIST INCLUDE ILACBL11----- 0017
LIST PHASE FCOBOL20,PHOCOPY(FCOBOL) 0018
LIST INCLUDE ILACBL20 0019
8 LIST PHASE FCOBOL22,PHOCOPY(FCOBOL)----- 0020
LIST INCLUDE ILACBL22 0021
LIST PHASE FCOBOL21,PHOCOPY(FCOBOL) 0022
9 LIST INCLUDE ILACBL21----- 0023
LIST PHASE FCOBOL25,PHOCOPY(FCOBOL) 0024
LIST INCLUDE ILACBL25 0025
0 LIST PHASE FCOBOL30,PHOCOPY(FCOBOL)----- 0026
LIST INCLUDE ILACBL30 0027
LIST PHASE FCOBOL40,PHOTBDIC(FCOBOL) 0028
11 LIST INCLUDE ILACBL40----- 0029
LIST REP 007B12 0014000 0953
LIST PHASE FCOBOL50,PHOTBDIC(FCOBOL) 0030
12 LIST INCLUDE ILACBL50----- 0031
LIST PHASE FCOBOL51,PHOTBDIC(FCOBOL) 0032
LIST INCLUDE ILACBL51 0033
3 LIST PHASE FCOBOL60,PHOTBDIC(FCOBOL)----- 0034
LIST INCLUDE ILACBL60 0035
LIST PHASE FCOBOL62,PHOTBDIC(FCOBOL) 0036
14 LIST INCLUDE ILACBL62----- 0037
LIST PHASE FCOBOL63,PHOTBDIC(FCOBOL) 0038
LIST INCLUDE ILACBL63 0039
5 LIST PHASE FCOBOL64,PHOTBDIC(FCOBOL)----- 0040
LIST INCLUDE ILACBL64 0041
LIST PHASE FCOBOL65,PHOTBDIC(FCOBOL) 0042
16 LIST INCLUDE ILACBL65----- 0043
LIST PHASE FCOBOL61,PHOTBDIC(FCOBOL) 0044
LIST INCLUDE ILACBL61 0045
17 LIST PHASE FCOBOL70,PHOTBDIC(FCOBOL)----- 0046
LIST INCLUDE ILACBL70 0047
LIST PHASE FCOBOL80,FCOBOL 0048
18 LIST INCLUDE ILACBL80----- 0049
LIST INCLUDE ILACBL81 0050
LIST INCLUDE ILACBL82 0051
1 LIST INCLUDE ILACBL83----- 0052
LIST INCLUDE ILACBL84 0053
LIST INCLUDE ILACBL85 0054
2 LIST INCLUDE ILACBL86----- 0055
LIST INCLUDE ILACBL87 0056
LIST INCLUDE ILACBL88 0057
3 LIST INCLUDE ILACBL89----- 0058
LIST INCLUDE ILACBL8A 0059
LIST INCLUDE ILACBL8B 0060
4 LIST INCLUDE ILACBL8C----- 0061
LIST INCLUDE ILACBL8D 0062
LIST AUTOLINK IJJCPDIN
5 LIST ENTRY-----

```

6

PHASE	YEP-AD	LOCDEF	HTCORE	DSK-AD	FSC TYPE	LABEL	LOADED	REL-FR		
FC08L	075810	070878	085837	021 03 01	CSECT	ILACBL00	070878	070878	RELOCATABLE	} ILACBL00 (Phase 00)
					CSECT	TLACBL01	081088	081088		
					CSECT	PHOCSECT2	07F81C	07D878		
					* ENTRY	START	07F81C			
					CSECT	IJGWZNZU	07F058	07D878		
					ENTRY	IJGWZNZZ	07F058			
					* ENTRY	IJGW77Z	07F058			
					* ENTRY	IJGW7RZ7	07F058			
					* ENTRY	IJGWZ7ZU	07F058			
					* ENTRY	IJGWZRZU	07F058			
					CSECT	IJJCPD0	07F4E8	07D878		
					ENTRY	IJJCPD0N	07F4E8			
					* ENTRY	IJJCPD1	07F4E8			
					ENTRY	IJJCPD2	07F4E8			
					ENTRY	IJJCPD3	07F4E8			
					* ENTRY	IJJCPD1N	07F4E8			
					CSECT	PHOTRST1	07F843	07D878		
CSECT	TBDATA	080318	07D878							
CSECT	PHOTPDIC	080710	07D878							
CSECT	PHOCOPY	080AEC	07D878							
CSECT	CLTR	080D00	07D878							
CSECT	PHOEND	081080	07D878							
CSECT	DATA	0820A0	081088							
CSECT	IJ2M0044	082960	081088							
CSECT	IJ5WZNZZ	082980	081088							
CSECT	MOBLEN	082F30	081088							
CSECT	ILAC02	082F38	081088							
FC08L05	080014	080010	084300	021 08 06	CSECT	TLACPL05	080000	080000	RELOCATABLE	ILACBL05
FC08L06	080014	080000	081004	021 08 02	CSECT	ILACPL06	080000	080000	RELOCATABLE	ILACBL06
FC08L07	080014	080000	082117	021 08 01	CSECT	ILACPL07	080000	080000	RELOCATABLE	ILACBL07
FC08L08	080014	080000	083097	021 08 01	CSECT	ILACPL08	080000	080000	RELOCATABLE	ILACBL08

PHASE	XBH-AD	LOCDEF	HICDEF	DSK-AD	ESD TYPE	LADEF	LOADDEF	REL-PR	
FCDEFBL10	080010	080010	080010	021 00 01	CSECT	TLA101	080000	080000	RELOCATABLE
					* ENTRY	PH1A	080012		
					CSECT	TLA102	081A08	080000	
					CSECT	TLA103	082480	080000	
					CSECT	TLA104	082FE8	080000	
					CSECT	TLA105	083E48	080000	
					CSECT	TLA106	084840	080000	
					CSECT	TLA107	0850C0	080000	
					CSECT	TLA108	085180	080000	
					CSECT	TLA1082	0850C0	080000	
					CSECT	TLA109	0861E0	080000	
					CSECT	TLA109A	0866D8	080000	
					CSECT	TLA110	087468	080000	
					CSECT	TLA111	087C90	080000	
CSECT	TLA112	087B50	080000						
FCDEFBL12	080010	080010	080010	022 00 01	CSECT	TLA101	080000	080000	RELOCATABLE
					* ENTRY	PH1A	080012		
					CSECT	TLA102	081810	080000	
					CSECT	TLA103	0823A8	080000	
					CSECT	TLA104	082130	080000	
					CSECT	TLA105	083E70	080000	
					CSECT	TLA106	084700	080000	
					CSECT	TLA107	084E10	080000	
					CSECT	TLA108	085000	080000	
					CSECT	TLA109	085258	080000	
					CSECT	TLA110	085F00	080000	
					CSECT	TLA111	086010	080000	
					CSECT	TLA111A	0879F8	080000	
					CSECT	TLA112	087B68	080000	
CSECT	TLA113	0886C8	080000						
CSECT	TLA114	088A98	080000						

ILACBL10  
(Phase 10)

ILACBL12  
(Phase 12)

PHASE	XED-AD	LOC-EE	HICORP	DSK-AD	ESD TYPE	LABEL	LOADED	PFL-ER	
FC09BL11	081500	080100	087833	022 05 03	CSECT	ILA100	080000	080000	RELOCATABLE
					CSECT	ILA101	081580	080000	
					* ENTRY	PH1F	081580		
					CSECT	ILA102	082170	080000	
					CSECT	ILA103	082418	080000	
					CSECT	ILA104	083508	080000	
					CSECT	ILA105	084108	080000	
					CSECT	ILA106	084830	080000	
					CSECT	ILA107	085418	080000	
					CSECT	ILA108	086278	080000	
					CSECT	ILA109	087140	080000	
CSECT	ILA10A	087100	080000						
FC09BL20	080AF8	080AF0	08617F	022 0A 01	CSECT	ILACBL20	080AE0	080AF0	RELOCATABLE
					ESECT	ILA201	0804F8	080AF0	
					CSECT	ILA210	080FC0	080AF0	
					CSECT	ILA202	081C70	080AF0	
					CSECT	ILA203	082AA8	080AF0	
					CSECT	ILA204	0834D0	080AF0	
					CSECT	ILA205	0838B8	080AF0	
					CSECT	ILA206	083CF8	080AF0	
					CSECT	ILA207	0840C8	080AF0	
					CSECT	ILA208	085090	080AF0	
					CSECT	ILA209	085D98	080AF0	
FC09BL22	081B38	080AF0	08607F	022 0D 05	CSECT	ILACBL22	080AF0	080AF0	RELOCATABLE
					CSECT	ILA202	081B28	080AF0	
					CSECT	ILA203	0826A0	080AF0	
					CSECT	ILA204	082BD0	080AF0	
					CSECT	ILA205	083A18	080AF0	
					CSECT	ILA206	084488	080AF0	
					CSECT	ILA207	084F98	080AF0	
					CSECT	ILA208	085B38	080AF0	
					CSECT	ILA209	086998	080AF0	

ILACBL11  
(Phase 11)

ILACBL20  
(Phase 20)

ILACBL22  
(Phase 22)



PHASE	XFR-AD	LOC-BS	MICROF	DEF-AD	ESD-TYPE	LABEL	LOCATED	REL-FR	
FC080L21	001360	080AF0	0096AE	022 12 01	CSECT		080AF0	080AF0	RELOCATABLE
					CSECT	3E0F42	080AF0	080AF0	
					CSECT	PHASE20	081360	080AF0	
					* ENTRY	PH20	081360		
					CSECT	IV5202	081538	080AF0	
					CSECT	IV5203	082148	080AF0	
					CSECT	IV5204	082300	080AF0	
					CSECT	IL2001	082508	080AF0	
					CSECT	IL2002	083150	080AF0	
					CSECT	IL2003	083730	080AF0	
					CSECT	IL2004	084278	080AF0	
					CSECT	IL2005	0840A0	080AF0	
					CSECT	IL2006	085A70	080AF0	
					CSECT	IL2007	086900	080AF0	
					CSECT	DEPMODE	086D40	080AF0	
CSECT	BURGEN	087D48	080AF0						
CSECT	PH200N	0887D0	080AF0						
FC080L25	081210	080AF0	081004	023 04 01	CSECT	IL2001	080AF0	080AF0	RELOCATABLE
					* ENTRY	PHASE25	081210		
					CSECT	IL2002	081540	080AF0	
CSECT	IL2003	081080	080AF0						
FC080L30	082008	080AF0	083F58	023 05 01	CSECT	TLACBL30	080AF0	080AF0	RELOCATABLE
					CSECT	TE0301	081210	080AF0	
					CSECT	TE0302A	081F78	080AF0	
					CSECT	TE0302	082808	080AF0	
					CSECT	TE0303	083810	080AF0	

ILACBL21  
(Phase 21)

ILACBL25  
(Phase 25)

ILACBL30  
(Phase 30)

PHASE	REF-AD	LOC-REF	HIC-REF	DSK-AD	ESP-TYPE	LABEL	LOAD-ED	REL-REF	
FCRBL40	081740	080710	0873AF	023 07 03	CSECT	ILACPL4A	080710	080710	RELOCATABLE
					CSECT	ILACPL4B	081508	080710	
					CSECT	ILACPL4C	082530	080710	
					CSECT	ILACPL4D	083530	080710	
					CSECT	ILACPL4E	084488	080710	
					CSECT	ILACPL4G	084880	080710	
					CSECT	ILACPL47	085098	080710	
					CSECT	ILACPL48	085F38	080710	
					CSECT	ILACPL4F	0864A8	080710	
					CSECT	ILACPL4G	0872F8	080710	
					CSECT	ILACPL4S	087FA8	080710	
					*-ENTRY-	PHN11	088740		
					CSECT	ILACPL4H	088520	080710	
CSECT	ILACPL4J	089610	080710						
CSECT	ILACPL4K	08A480	080710						
FCRBL50	081100	080710	08A0AF	023 08 05	CSECT	IEQ501	080710	080710	RELOCATABLE
					CSECT	IEQ502	0816F8	080710	
					CSECT	IEQ503	082128	080710	
					CSECT	IEQ505	083DA0	080710	
					CSECT	IEQ50C	0848A8	080710	
					CSECT	IEQ50G	0853F0	080710	
					CSECT	IEQ50D	086278	080710	
					CSECT	IEQ50E	086F70	080710	
					CSECT	IEQ50F	087C90	080710	
					CSECT	IEQ504	088970	080710	
*-ENTRY-	PHASE5A	089100							
CSECT	IEQ505	089370	080710						

ILACBL40  
(Phase 40)

ILACBL50  
(Phase 50)

PHASE	XFR-AD	LOCORE	HICORE	DSK-AD	ESD TYPE	LABEL	LOADED	REL-FR	
FC0B0L51	089140	080710	08A196	024 01 03	CSECT	IF0501	080710	080710	RELOCATABLE
3					CSECT	IF0505	081508	080710	
4					CSECT	IF0503	081748	080710	
					CSECT	IF050A	082728	080710	
5					CSECT	IF050E	082808	080710	
					CSECT	IF050G	083570	080710	
6					CSECT	IF050R	084148	080710	
7					CSECT	IF050H	084460	080710	
					CSECT	IF050P	085368	080710	
8					CSECT	IF050Q	086120	080710	
					CSECT	IF050J	086888	080710	
9					CSECT	IF050M	0875F8	080710	
					CSECT	IF050V	087D08	080710	
1					CSECT	IF0504	089130	080710	
					CSECT	IF050B	088A80	080710	
2					* - ENTRY	PHASE5	089140		
					CSECT	IF0505	0892F0	080710	
3	FC0B0L60	080724	080710	080598	024 08 01	CSECT	ILACPL60	080710	080710 RELOCATABLE
4					CSECT	ILA602	081368	080710	
					CSECT	ILA603	082298	080710	
5					CSECT	ILA604	083028	080710	
					CSECT	ILA605	083E98	080710	
6					CSECT	ILA605A	084580	080710	
					CSECT	ILA606	085598	080710	
7					CSECT	ILA607	086408	080710	
					CSECT	ILA608	087790	080710	

ILACBL51  
(Phase 51)

ILACBL60  
(Phase 60)

PHASE	XFR-AD	LOCORE	HICORE	DSK-AD	ESD TYPE	LABEL	LOADED	REL-FR	
FC0P0L62	080724	080710	0848F7	024 00 04	CSECT	ILACPL62	080710	080710	RELOCATABLE
					CSECT	ILA622	081278	080710	
					CSECT	ILA625	081F58	080710	
					CSECT	ILA622A	082A40	080710	
					CSECT	ILA623	082D70	080710	
					CSECT	ILA626	083C90	080710	
					CSECT	ILA627	084138	080710	
					CSECT	ILA629	0846C8	080710	
FC0P0L63	080724	080710	082876	024 10 03	CSECT	TLACBL63	080710	080710	RELOCATABLE
					CSECT	ILA631	0815D0	080710	
					CSECT	ILA632	082570	080710	
FC0P0L64	080724	080710	085E9F	024 11 06	CSECT	TLACBL64	080710	080710	RELOCATABLE
					CSECT	TLA643	080A28	080710	
					CSECT	TLA644	0818A0	080710	
					CSECT	TLA645	0826C0	080710	
					CSECT	TLA6455	083410	080710	
					CSECT	TLA646	083DF8	080710	
FC0P0L65	080722	080710	081C89	025 01 05	CSECT	TLA651	080710	080710	RELOCATABLE
					* ENTRY	PHASE65	080722		
					CSECT	ILA652	081168	080710	
					CSECT	ILA653	081590	080710	
					CSECT	TLA654	081C89	080710	
FC0P0L61	080722	080710	0825FF	025 02 05	CSECT	TLA6101	080710	080710	RELOCATABLE
					CSECT	TLA6103	081380	080710	
					CSECT	TLA6162	081790	080710	

ILACBL62  
(Phase 62)

ILACBL63  
(Phase 63)

ILACBL64  
(Phase 64)

ILACBL65  
(Phase 65)

ILACBL61  
(Phase 61)

PHASE	XFR-AC	LOGREF	FICREF	DSK-AC	ESD TYPE	LABEL	LOADED	REL-ER		
FC06DL70	07F85C	07F848	08B6AF	02F 04 01	CSECT		07F848	07F848	RELOCATABLE	} ILACBL70 (Phase 701)
					CSECT	ILACBL70	07F848	07F848		
					CSECT	TLA702	080660	07F848		
					CSECT	TLA712	081430	07F848		
					CSECT	TLA711	080900	07F848		
					CSECT	TLA700	080660	07F848		
					CSECT	TLA719	08AA70	07F848		
					CSECT	ACC	08AD18	07F848		
FC06DL80	07D870	07D878	08497E	025 00 01	CSECT	FC06DL80	07D878	07D878	RELOCATABLE	ILACBL80
					CSECT	INSCAN	07D880	07D880		ILACBL81
					CSECT	ENVSCAN	080000	080000		ILACBL82
					CSECT	DATASCAN	080C60	080C60		ILACBL83
					CSECT	PROSCAN	081AF8	081AF8		ILACBL84
					CSECT	MSGWRITE	084080	084080		ILACBL85
					ENTRY	FCFINPUT	084130			
					ENTRY	FCFQUEUE	084300			
					CSECT	FIPSVT	07D9D0	07D9D0		ILACBL86
					CSECT	CHKCOPY	07F120	07F120		ILACBL87
					CSECT	CHKCLBLS	07F228	07F228		ILACBL88
					CSECT	GETWORD	07F6B8	07F6B8		ILACBL89
					CSECT	GETLINE	07F430	07F430		ILACBL8A
CSECT	PUTLINE	07F5F8	07F5F8		ILACBL8B					
CSECT	MSGHNDLR	07FA18	07FA18		ILACBL8C					
CSECT	VERBCHK	07FC68	07FC68		ILACBL8D					
CSECT	IJJCPDIN	0845E8	0845E8							
ENTRY	IJJCPD3	0845F8								



SECTION 5. DATA AREAS

COMMUNICATION REGION

This chapter lists and describes the different cells that form the Communications Region (COMMON). COMMON is resident in storage throughout compilation as part of phase 00. Its format is defined as DSECTS in the rest of the phases, and therefore each phase can refer to any cell in COMMON by name. Much of the information saved in COMMON by phases 10 through 51 is used by phase 60 or phase 62 to form the Task Global Table (TGT) and Program Global Table (PGT) of the object program. The TGT and PGT are described in "Appendix B. Object Module."

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>
		<u>Hex</u>	<u>Decimal</u>	
COS	12	000	0	Phase 00 initialization coding.
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	4	12C	300	Address constants of TAMER used by the phases in table management requests.
AINSRT	each	130	304	
ADSTAT		134	308	
RELADD		138	312	
TAMNAD		13C	316	
ACCESW	1	140	320	ACCESS initialization switch (see "Appendix A. Table and Dictionary Handling"). X'01' if DICOT primed.
AMAINF	3	141	321	Pointer to the main free area for tables and the dictionary. This is also the address of the beginning of the HASH table. Routine ACCESS uses this field to locate the HASH table (see "Appendix A. Table and Dictionary Handling").
ALSTAM	4	144	324	Pointer to routine TBDICSPC, which obtains space for a new dictionary section (see "Appendix A. Table and Dictionary Handling").
LOCCTR	4	148	328	Contains the relative address of the next location available in the object program. It is initialized by phase 00 to the length of the INIT1 routine and incremented by phases 21 and 22 as they assign locations to data, and then by phase 60 or, under the optimizer version of the compiler, by phases 62, 63, and 64 as they assign locations to the global tables and procedure instructions.

<u>Cell</u>	<u>No. of</u>	<u>Displacement</u>		<u>Purpose</u>
<u>PROGID</u>	<u>Bytes</u>	<u>Hex</u>	<u>Decimal</u>	
	8	14C	332	PROGRAM-ID from the Identification Division of the source program. It is saved for use 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	Contains label information.
PRBLDISP	2	156	342	Contains displacement of beginning of PROCEDURE BLOCK CELLS in the PGT.
PNCTF	2	15B	344	Used in phase 11 as a counter for assigning unique PN numbers to source program procedure-names. In phase 60 or 62, it is set to the displacement of the PN field from the beginning of the PGT.
GNCTR	2	15A	346	Used in phases 10, 11, 22, 40, 50, and 51 as a counter for assigning unique GN numbers to compiler-generated procedure-names. In phase 60 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 60 or 62, it is set to the displacement of the VIRTUAL field from the beginning of the PGT. It is initialized to 1 by phase 00.
LITCTR	2	15E	350	Used in phases 50 and 51 as a counter to save the number of literals. In phase 60 or 62, it is set to the displacement of the LITERAL field from the beginning of the PGT.
WCMAX	2	160	352	Set by phase 50 to the size of the largest work area needed by any COBOL library subroutine. In phase 60 or 62, it is set to the displacement of the WORKING CELL field from the beginning of the TGT.
TSMAX	2	162	354	Set by phase 50 to the maximum number of doubleword cells needed for temporary storage at execution time by arithmetic statements. In phase 60 or 62, it is set to the displacement of the TEMPORARY STORAGE field 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>	
TS2MAX	2	164	356	Set by phase 51 to the number of bytes needed for temporary work areas by nonarithmetic statements. In phase 60 or 62, it is set to the displacement of the TEMPORARY 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 an item in Working-Storage or in a file containing an OCCURS clause with the DEPENDING ON option. 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 60 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 60 or 62 to allocate space for the CHECKPOINT CTR field of the TGT. Phase 60 or 62 sets it to the displacement of the CHECKPOINT 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). Intermediate and final values are stored in SBLIMX. In phase 60 or 62 it is set to the displacement of the SEL 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). Intermediate and final values are stored in VLCIMX. In phase 60 or 62 it is set to the displacement of the VLCI 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 60 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 11, and used by phase 70 in error message processing.
DICND2	4	174	372	Dictionary pointer for the last dictionary entry made in phase 22.
DICND1	4	178	376	Dictionary pointer for the last dictionary entry made in phase 11. If the UPSI feature was used, this cell contains instead the last dictionary entry made for an UPSI item by phase 22.

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>												
		<u>Hex</u>	<u>Decimal</u>													
WSDEF	7	17C	380	Set in phase 22 to the last seven bytes of the Data A-text element for the Working-Storage Section address definition, which gives the first base locator number and the length of the Working-Storage Section. When phase 60 assigns permanent base registers for base locators, it uses this information because it assigns base registers to the Working-Storage Section first. When OPT is specified, this field is not used since base registers are assigned differently.												
				<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 beginning of the Working-Storage Section</td> </tr> <tr> <td>WSSIZE</td> <td>3</td> <td>Number of bytes occupied by the Working-Storage Section</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 beginning of the Working-Storage Section	WSSIZE	3	Number of bytes occupied by the Working-Storage Section
<u>Name</u>	<u>Bytes</u>	<u>Meaning</u>														
WSSTRT	3	Starting address of working storage														
WSBL	1	BL number assigned to beginning of the Working-Storage Section														
WSSIZE	3	Number of bytes occupied by the Working-Storage Section														
ERRSEV	1	183	387	Set by phases 21 and 51 to the highest error severity level encountered in any phase.												
DICADR	4	184	388	ACCESS communication cell (see "Appendix A. Dictionary and Table 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 Save Area is needed at execution time. Used by phase 60 or 62 to determine whether that area should be set in the TGT and then set to the displacement of the REPORT SAVE field from the beginning of the TGT.												
SA2CTR	2	192	402	Used to save register 14 in declaratives for return.												
LCSECT	4	194	404	Contains the length of the object module CSECT.												
RGNCTR	2	198	408	Set by phase 60 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 60 or 62 and 64, and 70.												
PTYNO	1	19B	411	Priority number of current section.												

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>
		<u>Hex</u>	<u>Decimal</u>	
COMMAD	2	19C	412	Contains 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 is set by phase 10.
AMOVDC	4	1A0	416	Address of TAMER routine MOVDC. Phase 30 uses this cell.
SDSIZ	4	1A4	420	Set by phase 22 to the size of the largest SD entry in the program.
SEGLMT	1	1A8	424	Contains the 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 (hexadecimal 31). If phase 11 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 DATATAB. 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 60 or 62 to the displacement of the IND field from the beginning of the TGT.
IOPTRCTR	2	1AE	430	Number of input/output pointers resulting from SAME RECORD AREA clauses.
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 51 to the number of bytes needed for temporary storage by table handling verbs.
FLWSZ	1	1B4	509	Set by phase 01 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.
RPNCNTR	2	1B6	510	When OPT is specified, set by phase 51 to the number of PNs requiring an address constant cell in the PGT.
AGETALL	4	1B8	512	Address of routine GETALL in phase 00. This is used by phases 60 and 61 to obtain all available table space.
IDENTL	4	1BC	520	Set in phase 60 or 64 to the relative location of the first executable instruction.

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>
		<u>Hex</u>	<u>Decimal</u>	
BLCTR	2	1C0	448	Used in phase 22 as a counter for assigning unique identifying numbers to base locators for files and the Working-Storage Section. In phase 60 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 40 as a counter for assigning unique assign unique identifying numbers to variable procedure-names. In phase 60 or 62, it is set to four times the phase 40 value, which equals the number of bytes occupied by the VN cells.
ONCTR	2	1C4	452	Used in phase 51 as a counter to assign unique identifying numbers to ON control cells. In phase 60 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 40 as a counter to assign unique identifying numbers to PERFORM control cells. In phase 60 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 40 as a counter to assign unique identifying numbers to PERFORM save cells. In phase 60 or 62, it is set to the displacement of field PFMSAV from the beginning of the TGT.
XSACTR	2	1CA	458	Contains the 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 and then set to the total number of bytes needed for the Save Area. In phase 60 or 62, it is set to the displacement of field XSA from the beginning of the TGT. (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 60 or 62, it is set to the displacement of field XSASW from the beginning of the TGT.
PH6ERR	2	1CE	462	Used by phases 60, 62, 63, 64, and 65 to indicate that an error message is to be generated by phase 70. Bits 2, 4-7, and 9-10 are correlated to messages ILA6003I, ILA6005I-ILA6008I, and ILA6010I-ILA6011I. Phase 70 checks these bits and if a bit is set to 1, the corresponding message is generated.

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>
		<u>Hex</u>	<u>Decimal</u>	
RELLOC	4	1D0	464	Set in phase 60 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 60 or 62 to the length of the TGT.
VNILOC	2	1D6	470	Set in phase 60 or 62 to the relative location of the VNI field from the beginning of the PGT.
VNLOC	2	1D8	472	Set in phase 60 or 62 to the relative location of the VN field from the beginning of the TGT.
SUBCTR	2	1DA	474	Used in phase 40 as a counter to assign unique identifying numbers to subscripted references. In phase 60 or 62, it is set to the displacement of the field SUBADR from the beginning of the TGT.
PARMAX	2	1DC	476	Set in phase 51 to the size of the parameter area needed for parameter lists for macro instruction expansion of some of the source statements. In phase 60 or 62, it is set to the displacement of the PARAM field from the beginning of the TGT.
SPACING	1	1DE	478	Set by phase 01. Used for statistics.
PRBLNUM	1	1DF	479	Set by phase 62 to indicate the number of Procedure Block Cells in the PGT if the optimizer option (OPT) is specified. Phases 63 and 64 use this information.
CORESIZE	4	1E0	480	Set by phase 01. Used for statistics.
INDEX	4	1E4	484	Number of index names.
FIL5BUF	4	1E8	488	Used by phase 01 to store the address of SYS005 buffer. Used by phases 25 and 65.
ADATAB	4	1EC	492	Note address for first block of the DATATAB table on SYS005.
DATATBNM	2	1F0	496	Number of DATATAB blocks on SYS005.
OBODOTBN	2	1F2	498	Total number of bytes used for OBODOTAB entries on SYS005, including the slack bytes needed to align each OBODOTAB entry on a fullword boundary.
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 (FIBs). Phase 60 or 62 sets the field to the displacement of the FIB field 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>																			
FSTCDNM1	2	1FA	506	Number of CCB cells.																		
SWITV2	1	1FC	508	Switch																		
				<table border="1"> <thead> <tr> <th><u>Name</u></th> <th><u>Bit</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>COMPAT</td> <td>0</td> <td>ANS Version 2</td> </tr> <tr> <td>MRGBIT</td> <td>1</td> <td>MERGE specified</td> </tr> <tr> <td>CNTFDECL</td> <td>4</td> <td>Phase 11 found a declaratives card.</td> </tr> </tbody> </table>	<u>Name</u>	<u>Bit</u>	<u>Meaning</u>	COMPAT	0	ANS Version 2	MRGBIT	1	MERGE specified	CNTFDECL	4	Phase 11 found a declaratives card.						
<u>Name</u>	<u>Bit</u>	<u>Meaning</u>																				
COMPAT	0	ANS Version 2																				
MRGBIT	1	MERGE specified																				
CNTFDECL	4	Phase 11 found a declaratives card.																				
	3	1FD	509	Unused																		
TMCNTBSZ	4	200	512	Size of timer count table.																		
	2	204	516	Reserved																		
CCBLOC	2	206	518	Displacement of first CCB cell.																		
AMILOC	2	208	520	Reserved																		
INTVIRT	2	20A	522	Initial routine virtual number.																		
LOCTMCTT	4	20C	524	Start of timer count table.																		
	2	210	528	Unused																		
LISTERSW	1	212	530	Lister option switch.																		
				<table border="1"> <thead> <tr> <th><u>Name</u></th> <th><u>Bit</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>LSTRDECK</td> <td>0</td> <td>Lister source deck</td> </tr> <tr> <td>LSTRCPCH</td> <td>1</td> <td>Lister copy deck</td> </tr> <tr> <td>LSTRCOMP</td> <td>2</td> <td>Lister and compile</td> </tr> <tr> <td>LSTRONLY</td> <td>3</td> <td>Lister only</td> </tr> <tr> <td>LSTRPRC2</td> <td>4</td> <td>Two-column Procedure Division listing</td> </tr> </tbody> </table>	<u>Name</u>	<u>Bit</u>	<u>Meaning</u>	LSTRDECK	0	Lister source deck	LSTRCPCH	1	Lister copy deck	LSTRCOMP	2	Lister and compile	LSTRONLY	3	Lister only	LSTRPRC2	4	Two-column Procedure Division listing
<u>Name</u>	<u>Bit</u>	<u>Meaning</u>																				
LSTRDECK	0	Lister source deck																				
LSTRCPCH	1	Lister copy deck																				
LSTRCOMP	2	Lister and compile																				
LSTRONLY	3	Lister only																				
LSTRPRC2	4	Two-column Procedure Division listing																				
	45	213	531	Unused																		
PMAPADR	4	240	576	COBOL entry address.																		
BUFSIZE	4	244	580	Size of buffers for compilation work files. Used for statistics.																		
DATE	8	248	584	Set by phase 01 to the date of compilation.																		
TIME	8	250	592	Set by phase 01 to the time of the start of compilation.																		
PH7LOD	4	258	600																			
CRDNUMXX	2	25E	606	Used by phases 62 and 64 in sequencing the object deck.																		
DTFNUM	1	25F	607	Used in phase 21 as a counter for assigning unique identifying numbers to DTFs.																		
SDTFCTR	2	260	608	Used in phase 21 as a counter for assigning unique identifying numbers to secondary DTFs.																		
DTFNOXX	2	262	610	Used by phases 62 and 64 to calculate the relative address of DTF cells 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>	
DNCNT	4	264	612	Number of data-names for statistics. Set by phase 10.
VRBCNT	4	268	616	Number of verbs for statistics. Set by phase 11.
CURCRD	2	26C	620	Contains the compiler-generated card number of the text item currently being processed. If the text item is a verb, the high-order bit of CURCRD is on.





<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement Hex</u>	<u>Displacement Decimal</u>	<u>Purpose</u>
SWITCH	2	26E	622	Contains TRACE, DEBUG, SYMDMP, and Q-routine information.

<u>Name</u>	<u>Bit</u>	<u>Meaning</u>
SWTRCF	0	Set by phase 11 if TRACE is encountered so that phase 40 will generate TRACE coding at each procedure-name definition.
DELDSW	1	Set by phase 11 if there is a DISPLAY on SYSLST in a LABEL declarative. Tested by phase 51 to determine if a call to subroutine ILBDASY0 should be generated.
DSLDSW	2	Set by phase 11 if there is a DISPLAY on SYSPCH in a LABEL declarative. Tested by phase 51 to determine if a call to subroutine ILBDASY0 should be generated.
SPILL	3	Used if SYMDMP needs note or point on SYS005.
ALDSW	4	Set by phase 11 if there is an ACCEPT in a LABEL declarative. Tested by phase 51 to determine if a call to subroutine ILBDOSY0 should be generated.
MQVAR	5	Set by phase 22 if it builds a QVAR table.
LDECLSW	6	Set by phase 11 if there is a LABEL declarative.
MQFILE	7	Set by phase 22 if it builds a QFILE table.
SYMIPP	8	Set by phase 25 if there is an internal floating-point data item and SYMDMP is requested. Tested by phase 60 to determine if a virtual for subroutine ILBDTEF3 should be generated.
SYS5TD	9	Set by phase 01 if SYS005 is on tape. Tested by phase 65 to determine if a WRITE UPDATE can be done on SYS005 if it is a disk file. If SYS005 is a tape file, copy SYS005 information on SYS002 and then recopy on SYS005.
SORTRTN	10	Set by phase 30 if the sort RETURN verb is specified, and used by phase 51.
RERUNN	11	Set by phase 10 if RERUN is specified, and used by phases 21, 51, and either 60 or 62 and 64.
SORTSW	12	Set by phase 10 if SORT is specified.

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>																		
		<u>Hex</u>	<u>Decimal</u>																			
				<table border="1"> <thead> <tr> <th><u>Name</u></th> <th><u>Bit</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>NOFITSW</td> <td>13</td> <td>Phase 70 must read SYS003 for E-text since FRRTBL exceeded 256 bytes and has to be spilled.</td> </tr> <tr> <td>DOPH7</td> <td>14</td> <td>Tested by phase 60 or 62 or 64 to determine whether to call phase 70.</td> </tr> <tr> <td>RDERRFIL</td> <td>15</td> <td>Set by phase 60 or 64 if it did no processing. Tested by phase 70 to determine if SYS004 is to be read to find E-text.</td> </tr> </tbody> </table>	<u>Name</u>	<u>Bit</u>	<u>Meaning</u>	NOFITSW	13	Phase 70 must read SYS003 for E-text since FRRTBL exceeded 256 bytes and has to be spilled.	DOPH7	14	Tested by phase 60 or 62 or 64 to determine whether to call phase 70.	RDERRFIL	15	Set by phase 60 or 64 if it did no processing. Tested by phase 70 to determine if SYS004 is to be read to find E-text.						
<u>Name</u>	<u>Bit</u>	<u>Meaning</u>																				
NOFITSW	13	Phase 70 must read SYS003 for E-text since FRRTBL exceeded 256 bytes and has to be spilled.																				
DOPH7	14	Tested by phase 60 or 62 or 64 to determine whether to call phase 70.																				
RDERRFIL	15	Set by phase 60 or 64 if it did no processing. Tested by phase 70 to determine if SYS004 is to be read to find E-text.																				
PHZSW	1	270	624	Set by phase 01 from the compilation options. If the bit is on, the option was chosen.																		
				<table border="1"> <thead> <tr> <th><u>Name</u></th> <th><u>Bit</u></th> </tr> </thead> <tbody> <tr><td>LIST</td><td>0</td></tr> <tr><td>LISTX</td><td>1</td></tr> <tr><td>DFCK</td><td>2</td></tr> <tr><td>LINK</td><td>3</td></tr> <tr><td>SEQ</td><td>4</td></tr> <tr><td>FLAGW</td><td>5</td></tr> <tr><td>LIER</td><td>6</td></tr> <tr><td>ERRS</td><td>7</td></tr> </tbody> </table>	<u>Name</u>	<u>Bit</u>	LIST	0	LISTX	1	DFCK	2	LINK	3	SEQ	4	FLAGW	5	LIER	6	ERRS	7
<u>Name</u>	<u>Bit</u>																					
LIST	0																					
LISTX	1																					
DFCK	2																					
LINK	3																					
SEQ	4																					
FLAGW	5																					
LIER	6																					
ERRS	7																					
PHZSW1	1	271	625	Same as PHZSW for additional options. Phase 10 sets RPTWR on if a Report Section is encountered.																		
				<table border="1"> <thead> <tr> <th><u>Name</u></th> <th><u>Bit</u></th> </tr> </thead> <tbody> <tr><td>XREF</td><td>0</td></tr> <tr><td>CLIST</td><td>1</td></tr> <tr><td>SYM</td><td>2</td></tr> <tr><td>FLOW</td><td>3</td></tr> <tr><td>RPTWR</td><td>4 (Not a compiler option, this bit is tested to determine whether to load phase 12).</td></tr> <tr><td>APOST</td><td>5</td></tr> <tr><td>MAPSP</td><td>6 (SUPMAP)</td></tr> <tr><td>TRUNC</td><td>7</td></tr> </tbody> </table>	<u>Name</u>	<u>Bit</u>	XREF	0	CLIST	1	SYM	2	FLOW	3	RPTWR	4 (Not a compiler option, this bit is tested to determine whether to load phase 12).	APOST	5	MAPSP	6 (SUPMAP)	TRUNC	7
<u>Name</u>	<u>Bit</u>																					
XREF	0																					
CLIST	1																					
SYM	2																					
FLOW	3																					
RPTWR	4 (Not a compiler option, this bit is tested to determine whether to load phase 12).																					
APOST	5																					
MAPSP	6 (SUPMAP)																					
TRUNC	7																					
PHZSW2	1	272	626	Same as PHZSW for additional options.																		
				<table border="1"> <thead> <tr> <th><u>Name</u></th> <th><u>Bit</u></th> </tr> </thead> <tbody> <tr><td>SXREF</td><td>0</td></tr> <tr><td>STXIT</td><td>1</td></tr> <tr><td>ZWB</td><td>2</td></tr> <tr><td>CATALR</td><td>3</td></tr> <tr><td>[unused]</td><td>4</td></tr> <tr><td>SYMCAN</td><td>5</td></tr> <tr><td>STATE</td><td>6</td></tr> <tr><td>SYMDMP</td><td>7</td></tr> </tbody> </table>	<u>Name</u>	<u>Bit</u>	SXREF	0	STXIT	1	ZWB	2	CATALR	3	[unused]	4	SYMCAN	5	STATE	6	SYMDMP	7
<u>Name</u>	<u>Bit</u>																					
SXREF	0																					
STXIT	1																					
ZWB	2																					
CATALR	3																					
[unused]	4																					
SYMCAN	5																					
STATE	6																					
SYMDMP	7																					
PHZSW3	1	273	627	Same as PHZSW for additional options.																		
				<table border="1"> <thead> <tr> <th><u>Name</u></th> <th><u>Bit</u></th> </tr> </thead> <tbody> <tr><td>OPT</td><td>0</td></tr> <tr><td>SYNTAX</td><td>2</td></tr> <tr><td>CSYNTAX</td><td>3</td></tr> <tr><td>VERBR</td><td>7</td></tr> </tbody> </table>	<u>Name</u>	<u>Bit</u>	OPT	0	SYNTAX	2	CSYNTAX	3	VERBR	7								
<u>Name</u>	<u>Bit</u>																					
OPT	0																					
SYNTAX	2																					
CSYNTAX	3																					
VERBR	7																					

<u>Cell</u>	<u>No. of Bytes</u>	<u>Displacement</u>		<u>Purpose</u>
		<u>Hex</u>	<u>Decimal</u>	
PHZSW4	1	274	628	Same as PHZSW for additional options.

<u>Name</u>	<u>Bit</u>
VERBSUM	0
VERBREF	1
COUNT	2
Reserved	3
LVL	4

PH1BYTE	1	275	629	Switch for phases 10, 11, 22, 21, 50, 51, 62, and 63.
---------	---	-----	-----	---

<u>Name</u>	<u>Bit</u>	<u>Meaning</u>
	0	Unused
ADRSYM	1	Set by phase 51 when SYMDMP is in effect and a call to the ILEDVMO0 subroutine has been generated. Phase 60 tests the bit and generates a call to the JLEDADRC subroutine if the bit is set to 1.
OPTDISP	2	Set by phase 50 if OPT is specified and a call to the ILBDDSP0 subroutine is to be generated. Phase 51 tests the bit and generates the call if the bit is set to 1. If the bit is set to 0, phase 51 generates the call to ILBDDSS0.
QRTN1PBL	3	Set to 1 by phase 62 if all Q-routine GNs are contained in 1 Procedure Block. If phase 63 finds this bit turned on, it does not generate load instructions of register 11 for branches to Q-routines.
S370IN	4	Set to 1 by phase 10 if object-computer paragraph specifies IBM-370. Used by phases 50 and 51 to generate System/370 instructions.
DBLBUFIS	5	Set to 1 by phase 22 if there is an ISAM file with no RLSERVE NO clause which is opened input or I-O. Phase 21 tests to determine what module name to use for ISAM files.
EOPPH1	6	Set to 1 if phase 10 encounters an end-of-file condition in the source program.
UPSIBT	7	Set to 1 if an UPSI clause is specified.

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>													
SWITCH1X	1	276		Compiler internal switch byte												
				<table border="1"> <thead> <tr> <th><u>Name</u></th> <th><u>Bit</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>RENAMON</td> <td>0</td> <td>Set to 1 by phase 22 if RENAMTB exists. Tested by phase 25.</td> </tr> <tr> <td>OCCTBON</td> <td>1</td> <td>Set by phase 22 if OCCTBL exists. Tested by phase 25.</td> </tr> <tr> <td>DICTSPIL</td> <td>2</td> <td>Set by phase 00 if dictionary spill occurs. Used for statistics.</td> </tr> </tbody> </table>	<u>Name</u>	<u>Bit</u>	<u>Meaning</u>	RENAMON	0	Set to 1 by phase 22 if RENAMTB exists. Tested by phase 25.	OCCTBON	1	Set by phase 22 if OCCTBL exists. Tested by phase 25.	DICTSPIL	2	Set by phase 00 if dictionary spill occurs. Used for statistics.
<u>Name</u>	<u>Bit</u>	<u>Meaning</u>														
RENAMON	0	Set to 1 by phase 22 if RENAMTB exists. Tested by phase 25.														
OCCTBON	1	Set by phase 22 if OCCTBL exists. Tested by phase 25.														
DICTSPIL	2	Set by phase 00 if dictionary spill occurs. Used for statistics.														
SRTREERUN	8	277	631	File-name and logical unit number of file used for SORT checkpoint.												
	1	27F	639	Unused												
IPRECS	4	280	640	Number of source cards for statistics. Set by phases 10, 12, 11. Used by phase 60 or 64.												
AHSEGMSG	4	284	644	Address constant of header line in phase 00. Used by phase 60 or 64.												
LNCOUNT	2	288	648	Number of lines per page of compilation listing. Set to 1 by phase 01; used for statistics.												
	2	28A	650	Unused												
ATPLENT	4	28C	652	Length and address of Phase Length table.												
	14	290	656	Unused												

TABLE FORMATS

This chapter contains descriptions of all tables handled by the TAMER routines of Phase 00, whether built, used, and released within a single phase or passed to a later one. These tables differ from others in that additional storage can be obtained for them and they occupy space in the TAMER area rather than within a phase. Their descriptions are not intended to be complete, but to provide a quick reference to which phases use a table and for what general purpose. More explanation is provided in the chapters on the individual phases.

The tables are arranged in alphabetical order. Each description includes the TIB number (where applicable) and the format of an entry in the table. The TIB is explained in "Appendix A. Table and

Dictionary Handling." The following notes apply to the format diagrams in this chapter:

- The top row of figures shows the number of bytes in the field.
- Shaded areas indicate optional fields or a series of similar fields.
- n = the total number of bytes to follow in the entry.
- c = the number of bytes in the following field.
- Individual notes, applying to particular fields, are numbered consecutively with the numbers encircled.

ASCTAB  
(TIB 10)

Purpose  
Store address constants of DTFs for associated files when 3525 device if specified in ASSIGN clause.

Entry Frequency  
One entry for each DTF for an associated file.

1	1	1	1	1	1	3
Process- ing byte	System unit (SYSnnn) for associ- ated file	Device code	Hopper select	Type of file	Type of associ- ation	Address of DTF
		③	④	①	②	

Phases Involved  
Phase 21 builds and uses this table for generating address constants for each DTF for an associated file.

3	2
Address of address constant in this DTF	Card number

①	<u>Code</u> <u>Meaning</u>
	00    R (READ file)
	01    P (PUNCH file)
	02    W or M (WRITE file)

③	<u>Code</u> <u>Device</u>
	80    3525
	40    2560

②	<u>Code</u> <u>Meaning</u>
	80    V (READ, PRINT)
	40    X (READ, PUNCH, PRINT)
	20    Y (READ, PUNCH)
	10    Z (PUNCH, PRINT)

④	<u>Code</u> <u>Meaning</u>
	80    Primary hopper
	40    Secondary hopper

BLTABL  
(TIB 27)

Purpose  
Associate DTF numbers with buffers.

Entry Frequency  
One entry for each PD in source program.

1	1	2	1
DTF number	Buffer area number	Adjustment factor	FLAG
	②		①

Phases Involved  
Phase 21 builds and uses when generating buffers to determine how to initialize the base locator for each file.

- ① Bits Meaning, if ON  
 0 VSAM file  
 1-6 (Not used)  
 7 File is single buffered and unblocked, or indexed sequential.

② X'01' for VSAM file

BLASGTBL  
(TIB 16)

Purpose  
Assign object-time permanently loaded registers.

Entry Frequency  
One entry for each register: 6-10.

1	1
Type cell	BL, BLL, or overflow number
①	

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 hex 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.

ELUSTBL  
(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 PERFORM (24) element. It fills in the block number during Procedure A-text processing upon reading the VNREF element which follows the C004 element at the PERFORM EXIT.  
Phase 63 uses this table to determine whether the GN return point (C005) is in the same block as the EXIT statement and thus which Procedure Block is contained in register 11 on return from the performed procedure.

BUFTAB  
(TIB 29)

Purpose  
Store adcons to buffers for files.

1	1	1	1	1
DTF number	DTF number	SAME AREA number	SAME RECORD AREA number	Flag byte
				①

Entry Frequency  
One entry for each address constant for an I/O area that must be filled into a DTF.

Phases Involved

Phase 21 builds and uses for buffer generation.

Bits	Contents <sup>1</sup>
0-1	Size, minus 1, of DTF field receiving new (generated) address
2-4	Alignment of generated area 000 = at doubleword 001 = at doubleword + 1 . . 111 = at doubleword + 7
5-7	Unique area number for individual area within a file
1	X'01' for VSAM file

3	2	2
Address of DTF field or FIB field	Adjustment value (with sign) for area address	Maximum area for this generation or DTF

- ① The address, relative to the beginning of the object module, of either the DTF field receiving the generated address or, for VSAM, the FIB field (IRECOBL) receiving the displacement in the TGT of the record's first BL cell.

CARDINDX  
(TIB 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

Entry Frequency

One entry for each program fragment and one entry for each non-contiguous section other than the first within a segment.

Phases Involved

Phase 65 builds this table while reading SYS002 and building the PROCTAB table.  
Phase 65 writes this table on SYS005 and COBOL library subroutines use this table to relate card numbers to entries in the PROCTAB table.

Bits	Contents
0-19	Card number
20-23	Verb number (verb number is always 0 or 1)



CKPTBL  
(TIB 8)

Purpose  
Save RERUN statement information from source program scan.

7	1	2	2	4
External- name or SYSxxx	SYS number	CKPT counter number	DTP number	Number of records

Entry Frequency  
One entry for each RERUN statement.

Phases Involved  
Phase 10 builds this table from RERUN statement in source program.  
Phase 21 adds DTP number and checkpoint counter number.  
Phase 51 uses to generate coding to count and test "integer" with RERUN file OPENS, READS and WRITES.

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 61 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 61 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 60 or, when OPT is specified, phase 62 builds this table while processing Optimization A-text.  
Phase 60 or phase 62 uses this table with CONTBL and LTLTBL tables 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 60 or, when OPT is specified, phase 62 builds this table while processing Optimization A-text. Phase 60 or phase 62 uses this table with CONDIS and LTTBL tables to eliminate duplicate non-DISPLAY literals.

CTLTBL  
(TIB 14)

Purpose  
Store information on control names to check validity and build routines using them.

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		

Entry Frequency  
One for each control name.

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, and RET-ROUT routines.

2	2	2	Variable
GN number for control footing	Unused	Size of previous entry	Control-name including qualifiers, indexes, and subscripts, if any, in P0-text form.

① In P0-text form:

Bytes	Contents
0	23
1	05
2	- (hyphen)
3-6	nnnn

② Bit                      Meaning, if on

0	Control-name is subscripted or indexed
1-2	Unused
3	Control footing specified
4-6	Unused
7	Control heading specified

CVIRTB  
(TIB 12)

<u>Purpose</u> Store virtual name from definition elements during virtual optimization.	8
<u>Entry Frequency</u> One entry for each unique virtual.	Virtual

Phases Involved  
Phase 60 or, when OPT is specified, phase 62 makes an entry when it finds a virtual definition during Optimization A-text processing.  
phase 60 or Phase 62 uses this table with the VIRPTR table to eliminate duplicate references to virtuals.

DATATBL

①

<u>Purpose</u> Store information for XREF or SXREF processing.	3	33	2
<u>Entry Frequency</u> One entry (DATA record) for each DEF-text element.	Pointer to dictionary entry for data-name or PN number	External name in EBCDIC, defining card number, and a variable number of card numbers referring to name	Ascending source order pointer (SXREF only)

Phases Involved  
Phase 61 builds and uses this table in producing an XREF or SXREF listing.

①

There is no TIB for this table. Phase 61 uses the Phase 00 routine GETALL to get space, but moves data in and out of the table by itself.

2	1	3	1	3
Descending source order pointer (SXREF only)	Offset in bytes from start of record to location for the next card number referring to name	Pointer to current (last) record	Length of external name	Pointer to first OVERFLOW record

DBGTBL  
(TIB 13)

<u>Purpose</u> Store information on procedure-names referred to in DEBUG statements.	2	2
<u>Entry Frequency</u> One entry for each procedure-name referred to by DEBUG.	PN number for procedure-name	GN number for debug procedure associated with procedure-name

Phases Involved  
Phase 40 builds this table from PNs in P1-text referred to in DEBUG statements and from GNs from GNCTR in COMMON.  
Phase 40 uses this table to issue P2-text CALL

statements to DEBUG procedures.

DEFSBS  
(TIB 18)

Purpose  
Store subscript-defining string until all subscripts in statement are collected.

1	Variable	1	Variable
	First element in string		Last element in string

Entry Frequency  
One entry for current string being built.

Phases Involved  
Phase 40 builds this table from P1-text of subscripted data-name.  
Phase 40 uses this table with STRING table to issue P2-text subscript strings.

DETTBL  
(TIB 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 ①	②	GN number for this detail report group	③	Displacement of entry in RWRTBL for report-name associated with this detail report group

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.  
Phase 11 uses this table to generate detail-names.

2	1	⑤
GN number for USM-ROUT	④	

- ① Left-justified, padded with binary zeros in low-order bytes.
- ② Length of the 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 Code for unique detail-name.

Code	Meaning
00	This entry was made as the result of a unique detail group data-name.
01	This entry is a non-unique detail group name that must be qualified.

⑤ First entry in 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 11 and 22 build as they build the dictionary.  
Phases 11, 22, 21, 25, and 30 use to find dictionary sections.

- |   |             |  |
|---|-------------|--|
| ① | <u>Bits</u> | <u>Meaning, If ON</u>  |
|   | 0           | Section has been spilled   |
|   | 1           | Section is now in main storage   |
|   | 2           | Section has been updated   |
|   | 3           | A section, which had been spilled and read back into storage, has been modified and the copy on the external device is obsolete. |
|   | 4-7         | Not used.  |

② Address on disk where section has been spilled, if it was ever spilled.

DRPTEL  
(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) |                         |

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.

ENVTBL  
(TIB 3)

Purpose  
Store file information from Environment Division to be merged with Data Division information to form Data IC-text.

2	7	1	2	1
Compiler-generated source card number	Either external name (if specified) or SYSnnn	nnn from SYSnnn portion of file-name in binary	Flag field (1)	(1a)

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

1	2	1	1
Buffer offset	Pointer to entry in PIOTBL (2)	Device organization codes for associated files (2a)	Flag field (3)

3	1	2	1	2
Unused	Number of SORT work units	Pointer to entry in CKPTBL (2)	Number assigned to SAME AREA clause	Flag field (4)

1	1	1	1	1	1	1	8	1
5	Number assigned to SAME RECORD AREA clause	Integer in MULTIPLE FILE...POSITION clause	Number of tracks for CYL-OVERFLOW	Device type code	Device number of highest index	Number assigned to SAME SORT AREA clause	(6a)	c
				(6)	(6)			

30	4	4	4	4	4
File-name in FD entry	TRACK AREA size	NOMINAL KEY and qualifiers	ACTUAL KEY and qualifiers	RECORD KEY and qualifiers	Pointer to data-name qualifiers from APPLY option (APPLY CORE-INDEX)
	(7)	(7)	(7)	(7)	

4	4	4	4
Pointer to dn qualifiers from APPLY option	File status and qualifiers	Password and qualifiers	Unused
	(7)	(7)	

- ①
- | Bit | Meaning  | Bits | Meaning  |
|-----|--|------|--|
| 0   | 1 = RANDOM ACCESS  | 1    | 1 = No organization parameter specified in system-name   |
| 1-3 | Organization<br>000 = Not specified<br>001 = INDEXED<br>010 = DIRECT<br>100 = RELATIVE           | 2-4  | ORGANIZATION clause  |
| 4-6 | Device Class<br>000 = Not specified<br>001 = DIRECT-ACCESS<br>010 = UNIT-RECORD<br>100 = UTILITY |      | <u>Code</u> <u>Meaning</u><br>000 Not specified<br>001 SEQUENTIAL<br>010 INDEXED<br>100 Reserved |
| 7   | 1 = No RESERVE ALTERNATE AREA  | 5    | ACCESS MODE IS DYNAMIC   |
| 8   | 1 = SELECT OPTIONAL  | 6    | Unused   |
| 9   | 1 = SAME AREA specified  | 7    | PASSWORD data-name specified with RECORD KEY or for the file.                                    |
| 10  | EXTENDED-SEARCH  |      |  |
| 11  | SAME RECORD AREA specified   |      |  |
| 12  | SAME SORT AREA specified   | ③    | <u>Bit</u> <u>Meaning</u><br>0 1 = COPY  |
| 13  | 1 = CKPTBL pointer exists  | 1    | Unused   |
| 14  | 1 = PIOTBL pointer exists  | 2    | RECORD CONTAINS clause   |
| 15  | 'ALTERNATE' specified in RESERVE clause  | 3-4  | BLOCK CONTAINS integer<br>00 = Not specified<br>01 = RECORDS<br>10 = CHARACTERS                  |
|     |  | 5-6  | LABEL RECORDS<br>00 = Not specified<br>01 = STANDARD<br>10 = OMITTED<br>11 = Data-name           |
|     |  | 7    | 1 = REPORT clause  |
- ② Displacement of entry in table.
- ②a Device organization codes for associated files
- | Bits | Meaning   |
|------|---|
| 0    | 1 = AS specified in ORGANIZATION parameter of system-name |

<u>Bit</u>	<u>Meaning</u>
0-1	TRACK AREA 00 = Not specified 10 = Integer
2	Unused
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	CYL-OVERFLOW
10	'integer' of RESERVED clause not in valid range
11	MULTIPLE REEL/UNIT
12	MULTIPLE FILE TAPE
13	MASTER-INDEX
14	CYL-INDEX
15	Unused

19	3881
20	Unused
21	5425R
22	5425P
23	5425W
24	Unused
25	2560R
26	2560P
27	2560W
28-30	Unused
31	2311
32	Unused
33	2314 or 2319
34	2321
35	3340
36	3540
37-39	Unused
40	2400, 3410, or 3420

<u>Bits</u>	<u>Contents</u>
0-3	Temporary storage for Phase 10, Phase 11, or the number of nonstandard reels.
4	CORE-INDEX
5-6	Unused
7	'integer' of ASSIGN clause not in valid range

<u>Byte</u>	<u>Meaning</u>
35	Count of ALTERNATE RECORD KEY clauses
36-37	INDTBL displacement, the displacement to the first entry associated with this file (that is, if any ALTERNATE RECORD KEY clauses appear under the select)

<u>Code</u>	<u>Device</u>
1	1442R
2	1442P
3	2520R
4	2520P
5	2540R
6	2540P
7	2501
8	1403, 5203, or 3203
9	1404
10	1443
11	1445
12	3211
13	3505 or 3504
14	3330
15	3525R
16	3525P
17	3525W or 3525M
18	Unused

<u>Bit</u>	<u>Meaning</u>
0	Incorrect class
1	Incorrect device
2	Incorrect organization parameter
3-7	Unused

<u>Bit</u>	<u>Meaning</u>
0	Assigned
1	Assigned
2-7	Unused

40-42 Unused

<u>Bytes</u>	<u>Contents</u>
0-1	Length of name(s) in bytes (or binary literal if TRACK AREA is specified)
2-3	Displacement of name in QNMTBL



ERRTEL  
(TIB 10)

Purpose  
Store E-text to separate it from Data A-text for phase 70.

8	1	1	Variable		1	1	Variable
E-text for basic message	00	c	First message parameter		00	c	Last message parameter

①

Entry Frequency  
One entry for each message to be generated.

Phases Involved  
Phase 60, or when OPT is specified, phase 64 build this table from Data A-text. Phase 70 uses this table to generate error messages.

① Table ends with one-byte element X'77'

FDTAB  
(TIB 28)

Purpose  
Pass record description information from phase 22 to 21.

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

Entry Frequency  
One entry for each PD in source program.

Phases Involved  
Phase 22 builds from Record Descriptions in ATF-text. Phase 21 uses to generate DTFs and buffers.

- | <p>①</p> <table border="0"> <tr> <th>Bits</th> <th>Use</th> <th></th> </tr> <tr> <td>0-3</td> <td>Number of base locators</td> <td>6</td> </tr> <tr> <td>4</td> <td>ODO2 switch: ON, if any record descriptions contained more than one ODO clause.</td> <td>7</td> </tr> <tr> <td>5</td> <td>ODO switch: ON, if any record description contained an ODO clause.</td> <td></td> </tr> </table> | Bits  | Use |  | 0-3 | Number of base locators | 6 | 4 | ODO2 switch: ON, if any record descriptions contained more than one ODO clause. | 7 | 5 | ODO switch: ON, if any record description contained an ODO clause. |  | <p>② Used by phase 21 to get dictionary attributes when LATRNM returns a duplicate code.</p> |
|--|---|-----|--|-----|-------------------------|---|---|---|---|---|--|--|--|
| Bits   | Use   |     |  |     |                         |   |   |   |   |   |  |  |  |
| 0-3  | Number of base locators   | 6   |  |     |                         |   |   |   |   |   |  |  |  |
| 4  | ODO2 switch: ON, if any record descriptions contained more than one ODO clause. | 7   |  |     |                         |   |   |   |   |   |  |  |  |
| 5  | ODO switch: ON, if any record description contained an ODO clause.              |     |  |     |                         |   |   |   |   |   |  |  |  |

FILTEL  
(TIB 2)

Purpose  
Store virtuals related to input/output label- or error-processing virtuals.

4	8
Location to be entered on ESD card	Name of virtual

Entry Frequency  
One entry for each virtual related to input/output label- or error-processing routines.

Phases Involved  
Phase 60, or when OPT is specified, phase 62 builds this table during virtual optimization. Phase 60 or phase 62 uses this table to place virtual names in DTFs, for example, for SYNAD routines.

FNTBL  
(TIB 10)

Purpose

Store Environment Division information about a file for Procedure Division processing.

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 trailer labels

Entry Frequency

One entry for each file.

Phases Involved

Phase 10 builds this table from the ENVTBL table and the Data Division.

Phase 12 uses in Report Section processing.

Phase 11 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 Bit	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.

4	2
①	Generated 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 four bytes 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

GNATBL  
 (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	GN first number for Q-routine call

Entry Frequency  
 One entry for each  
 GN number.

Phase Involved  
 Phase 51 builds this table  
 and uses it to generate  
 call to the Q-routines  
 after return from the  
 CALL statement.

GNPWDBTB  
 (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.

GNLBDTBL  
(TIB 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.

GNTBL  
(TIB 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 60 builds this table from GNCTR and PN and GN equate strings. Phase 60 uses this table to optimize procedure-names and process Procedure A-text.

GPLSTK  
(TIB 10)

Purpose

Count length of group item while subordinate items are being processed.

Entry Frequency

One entry for each group item being currently processed.

4	2	2	2
Flag bytes	OD2TBL displacement of entry for OCCURS DEPENDING ON object within group item (if none, field contains zeros)	Maximum number of occurrences (if none, field is 0)	Generated card number

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

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 of entry for VALUE clause literal in VALGRP table, if used (0, if not)

2	1	1
Displacement of entry for VALUE clause literal in VALTRU table, if used (0, if not)	Flag byte (2)	Flag byte (3)

- ①
- | Bits  | Meaning   |
|-------|---|
| 0     | 1 = group occurs more than once; alignment required |
| 1     | 1 = 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     | Unused  |
| 8-11  | Minor code (see LD dictionary entry format)         |
| 12-13 | MBS (must be subscripted) bits                      |
| 14    | 1 = Item contained OCCURS or an ODO clause          |
| 15-31 | Length of group or VLC                              |

- ② 

<u>Bits</u>	<u>Meaning</u>
3	Justified
4-7	Set to X'1111' if USAGE is other than DISPLAY

- ③ Bit 3 is set to 1 for the master of an ODO clause.

GVPNTBL  
(TIB 3)

Purpose

Store pointer to FNTBL for VSAM files referred to in USE AFTER STANDARD ERROR/EXCEPTION with GIVING option.

2
Displacement in FNTBL

Entry Frequency

On entry for each file-name mentioned in USE Declarative.

Phases Involved

Phases 11 builds the table using the FNTBL. Phase 11 uses the table to complete the FNTBL entries after building the GVNMTBL table.

GVNMTBL  
(TIB 4)

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 11 builds the table from the entries in the CURBCD and CURN data areas and for qualified data names from the QLTABL table. Phase 11 uses the table to create the OPEN coding for VSAM files.

INDKEY  
(TIB 31)

Purpose  
Store OCCURS...DEPENDING ON (ODO) information for use in table handling.

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 30 uses this table to process SEARCH and SEARCH ALL verbs.

1	3	1	3	1
n	Dictionary pointer to subject of OCCURS or maximum number of occurrences	Flag byte	Dictionary pointer to object of OCCURS or maximum number of occurrences	Number of index-names

3	3	1
Dictionary pointer to first index-name	Dictionary pointer to last index-name	Number of keys

3	3
Dictionary pointer to first key	Dictionary pointer to last key

① Contents of dictionary pointer:

Bits	Contents
0-1	Zeros
2-14	Dictionary section number
15-23	Displacement in section

②

Bit	Meaning
0-2	Unused
3	1 = Next three bytes contain pointer to subject of ODO clause 0 = Next three bytes contain maximum number of occurrences.
4-6	Unused
7	1 = Error detected in key processing (Phase 30 uses) 0 = No error found

INDXTB  
(TIB 27)

Purpose  
Save all index-names associated with a data item.

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.

1	1	Variable
04 (hex)	c	Index-name in EBCDIC

IND2TBL  
(TIB 35)

Purpose  
Store information on VSAM files for ORGANIZATION IS INDEXED.

3	3	1
Dictionary pointer for RECORD KEY	idk parameters for RECORD KEY	level number

Entry Frequency  
One entry for each PD 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.

IOPTBL  
(TIB 5)

Purpose  
Store SAME RECORD AREA information.

1	1
DTF number	IOPTR number

Entry Frequency  
One entry for each unique BL number within an output file PD entry that also contains a SAME RECORD AREA clause.

Phases Involved  
Phase 51 builds and uses as object-time I/O buffer pointers.

KEYTAB  
(TIB 26)

Purpose  
Save all key-names associated with a data item.

1	1	Variable
Flag	c	Key-name in EBCDIC
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	<u>Bits</u>	<u>Meaning, if on</u>
	0-5	Unused
	6	Descending key
	7	Ascending key



KEYTBL  
(TIB 20)

Purpose  
Ensure that, if key is tested in WHEN condition, all previous keys are also tested.

Entry Frequency  
One entry for each key in KEY clause associated with identifier-1 in SEARCH ALL statement.

Phases Involved  
Phase 40 builds this table while processing a SEARCH ALL statement.  
Phase 40 uses this table to make sure SEARCH ALL statement processing tests keys associated with identifier-1 and does not test a key without having tested the previous key.

3	1
Addressing parameters for item (IDK) from dictionary entry	
①	②

①	<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

② Initially 0, but set to 1 whenever a WHEN condition for KEY is found during processing of this SEARCH ALL statement.

LABTBL  
(TIB 13)

Purpose  
Save LABEL RECORD data-names referred to in a Data IC-text FD entry.

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.

1	Variable
c	LABEL RECORD data-name in EBCDIC

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 60 or, when OPT  
is specified, phase  
62 builds this table  
while building CCNDIS and  
CONTBL tables.  
Phase 60 or, when OPT  
is specified, phase  
64 uses this table  
with CONDIS and CONTBL  
tables to eliminate duplicate  
DISPLAY and non-DISPLAY  
literals.

MASTODO  
(TIB 13)

Purpose  
Identify masters of  
OCCURS...DEPENDING  
ON clause<sup>1</sup> if SYMDMP is  
specified.

3
Dictionary pointer for master of an ODO <sup>1</sup>

Entry Frequency  
One entry for each  
master of an OCCURS...  
DEPENDING ON clause.<sup>1</sup>

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<sup>1</sup> for the DATATAB  
table.

<sup>1</sup>See "Glossary for definition of "master of an  
OCCURS Clause with the DEPENDING ON Option."

NPTTBL  
(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 to objects  
and subjects of OCCURS...  
DEPENDING ON clauses.  
It is used to build the  
QFILE table.

2	2	2	2
DTF Number	①	①	X'FFFF"



Entry Frequency

One entry for each file whose record descriptions contain at least one object and/or subject of an OCCURS...DEPENDING ON clause.

Phases Involved

Phase 22 builds and uses 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.

OCCTBL  
(TIB 2)

Purpose

Store information about items in OCCURS and OCCURS...DEPENDING ON clauses if SYMDMP 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 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:

<u>Bits</u>	<u>Contents</u>
0-8	Relative block number within the OBODOTAB table
9-15	Displacement in fullwords within the block

ODOTBL  
(TIB 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 is specified and the program contains and 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 highest 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. Phase 25 uses this table in building the OBODOTAB table.

Variable	1	Variable	1	1
EBCDIC name of lowest-level qualifier	c	EBCDIC name of object	c	00

OPLQTB

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
Referencing card number	Referencing card number	Referencing card number	Referencing card number

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.

1	3
Offset	Pointer to preceding OVERFLOW record

Phases Involved  
 Phase 61 builds and uses this table to store referencing card numbers for the SXREF or the XREF option.

- ① There is no TIB for this table. Phase 61 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.

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 end-of-range

Entry Frequency  
 One entry for each delimiting procedure-name.

Phases Involved  
 Phase 40 builds this table from P1-text procedure-names and VNCTR in COMMON. Phase 40 uses this table to keep track of delimiters of performed procedures to set up return VNs.

PIOTBL  
(TIB 7)

Purpose  
Store input/output information for a file from Procedure Division.

3	1
Switch bytes	VSAM switch byte
①	②

Entry Frequency  
One entry for each file.

Phases Involved  
Phase 10 sets aside space for one entry for each file.  
Phases 11 and 12 set appropriate bits.  
Phase 21 uses this table to generate Data A-text.

①

Bits	<u>Statement referring to file, if bit is on</u>
0	OPEN INPUT
1	OPEN OUTPUT
2	OPEN I-O
3	Unused
4	WRITE AFTER ADVANCING
5	CLOSE WITH LOCK
6	CLOSE
7	REWRITE
8	RERUN
9	OPEN INPUT REVERSED
10	SEEK
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	Unused
22	REVERSED
23	Unused

②

Bit	<u>Statement referring to file if on</u>
0	KEY IS data-name (for START)
1	DELETE
2	USE AFTER STANDARD ERROR GIVING DN
3	OPEN EXTEND (physical sequential only)
4-7	Unused

PNATEL  
(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.

PNFWDETBL  
 (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.

PNLARTBL  
 (TIB 18)

Purpose  
 Determine inter-block and intra-block 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.

PNLBDTBL  
 (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 40 builds this table from P1-text scan.

Phase 40 uses this table with PSIGNT table to stack operands until the string is ready to be created.

① Number of bytes in the preceding field.

PNQTB

(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 11 builds this table from Procedure Division.

Phase 11 uses this table to complete procedure-name dictionary entries.

1	Variable
c	Procedure-name qualifier in EBCDIC

① Bit

Bit	Meaning, if on
0	Procedure-name
1	Section-name
2	Either name follows THRU in PERFORM...THRU, or follows 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 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 type of section.
12	Declarative error routine
13	Declarative label routine
14	Unused
15	Declarative report section.

PNTABL  
(TIB 5)

Purpose

Store information on references to unqualified PNs for later completion of PNs 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 11 builds this table from Procedure Division. Phase 11 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 follows 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 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 type of section.
12	Declarative error routine
13	Declarative label routine
14	Unused
15	Declarative report section

② Bit	Meaning, if on
0	Dictionary was searched 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 60 builds this table from PNCTR in COMMON and PN equate strings. Phase 60 uses this table to optimize PNs and process Procedure A-text.

PNUTBL  
(TIB 6)

Purpose  
Optimize procedure-names by eliminating those not referred to or, when OPT is specified, 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 60 uses this table to eliminate names not referred to, or, when OPT is specified, phase 62 uses this table to determine which PNs are entry points.

PROCTNDX  
(TIB 5)

Purpose  
Store information about the PROCTAB table which is written on SYS005 if SYMDMP option is in effect.

3	3	4
Card/verb number for first entry in PROCTAB block	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 SYS002 and building the PROCTAB table. Phase 65 writes this table on SYS005 and COBOL library subroutines use this table to address entries in the PROCTAB table.

①	<u>Bits</u>	<u>Contents</u>
	0-19	Card number
	20-23	Verb number

PSHTBL  
(TIB 17)

Purpose  
Store GN numbers of ELSE branches in nested and compound IF statements.

1	2
16 = Master label	GN
0 = Otherwise	number

Entry Frequency  
One entry for each false branch.

Phases Involved  
Phase 40 builds this table from P1-text and GNCTR in COMMON.  
Phase 40 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 statement being processed.

Phases Involved  
Phase 40 builds this table from P1-text.  
Phase 40 stacks operators until strings are ready to generate. Table 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 branch	GN number for bypass branch
0 = 'NOT' is not active		

Entry Frequency  
One entry for each decision in statement.

Phases Involved  
Phase 40 builds this table from P1-text compound IF statements and GNCTR in COMMON.  
Phase 40 uses this table with the PSHTBL table to select branches for P2-text.

P1BTBL  
(TIB 2)

Purpose

Pass phase operating information from phase 10 to phase 11 or 12 or from phase 12 to phase 11

Entry Frequency

One entry made at end of phase 10 processing and, optionally, at end of phase 12 processing.

2	8	8	3	80
n	Generated card number for last record read (packed)	Generated card number for last record read (unpacked)	Unused	Last record read

Phases Involved

Phases 10 and 12 build this table from stored information. Phase 11 and/or 12 moves data into its own data areas.

6	18	82	6	2
Sequence number of last record read	Error in-formation on record	INSERT card work area from BASIS	Sequence number from BASIS	Phase switches (1)

114	4	4	2
Unused	SYS004 pointer for BASIS/COPY READ	Unused	Phase switches (2)

2	4	80	8	2	1
Current generated card number	Phase switches (3)	Double-buffer for COPY... REPLACING	(4)	Contents of columns 72 and 73	Phase switches (5)

(1)

Bits	Byte 1	Byte 2
0	Unused	INDLSW
1	COPYSW	INSTSW
2	BASISW	INDEERR
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

(2)

Bits	Byte 1	Byte 2
0	FSTRC	Unused
1	Unused	Unused
2	Unused	Unused
3	Unused	USFPDL
4	Unused	USEPDE
5	Unused	SRTSW
6	Unused	Unused
7	Unused	Unused

(4) For Phase 10  
Bytes 0-2 = Report Writer generated name  
Bytes 3-7 = Not used

(5) Bits 0-6: Not used  
7: FRGNSW

QALTBL  
(TIB 23)

Purpose

Work table, for example, to store qualified names and qualifiers in reverse order of appearance in RD.

1 Variable	1 Variable
c Name in	c Name in
EBCDIC	EBCDIC

Entry Frequency

One for each name in the current string of a name and its qualifier.

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 all files in which at least one of the records contains an OCCURS... DEPENDING ON clause.

2	2
DTF number for file	GN number for single Q-routine for file or for chain of GNs, where there is more than one Q-routine associated with the file

①

Entry Frequency

One entry for each file of this type.

Phase Involved

Phase 22 builds this table from Q-routines for file. Phase 30 adds Q-routine GN numbers to dictionary attributes for corresponding files.

① 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

Entry Frequency

One entry for each OD2TBL table entry.

Phase Involved

Phase 22 builds this table from dictionary and OD2TBL table. Phase 22 combines this table with OD2TBL and QRTN tables to form QVAR table. Phase 25 uses this table with QRTN and OCCTBL to build the ODOTBL.

OLTABL  
(TIB 1)

Purpose

Store qualified names in order of receipt.

1	1	Variable	1	Variable	1	Variable	1
35	00	Name in EBCDIC	①	First Qualifier Name	①	Last qualifier name in EBCDIC	①

Entry Frequency

One table entry containing name currently being processed and its qualifiers.

Phases Involved

Phase 10 builds this table when a qualified EBCDIC name is inserted in the QNMTBL or OD2TBL table. Phase 12 builds and uses when a qualified EBCDIC name is to be written in P0-text. Phase 11 builds and uses when a qualified EBCDIC name is to be written in P0-text.

① Count of bytes in preceding field.

QNMTBL  
(TIB 2)

Purpose

Store BCD names for REPORT, LABEL RECORD, ACTUAL KEY, NOMINAL KEY, RECORD KEY, and APPLY CORE-INDEX clauses from Data and Environment Divisions until Data IC-text is written.

2	1	Variable	1
n	c	Last qualifier name in EBCDIC	c

Variable	1
Qualified name in EBCDIC	00

Entry Frequency

One entry for each name of this type.



Phases Involved

Phase 10 builds this table from clauses in PD, SELECT, and APPLY statements. Phase 10 uses this table to write out Data IC-text.

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 to follow	Displacement of OD2TBL entry for 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 table. Phase 22 combines this table with OD2TBL and QITBL to form QVAR table. Phase 25 uses this table with QITBL and OCCTBL to build ODOTBL.

QSEL  
(TIB 1)

Purpose

Hold information on secondary base locators (SBLs).

1
First SBL number associated with an INCRA verb

①

Entry Frequency

One entry for each INCRA verb generated.

Phases Involved

Phase 22 builds as it generates INCRA verbs for Q-routines. Phase 30 uses table to generate for each INCRA verb a P1-text literal the value of which equals the number of SRLs associated with the verb.

① Last entry is followed by a byte of zeros.

QTBL  
(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 60 or, when OPT is specified, phase 64 makes an entry when it finds a Q-routine identification element in Data A-text. Phase 60 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.

Entry Frequency

One entry for each item.

4	2	1
Pointer to dictionary entry for object of OCCURS DEPENDING ON for first item ①	GN number for Q-routine for first item	②

Phases Involved

Phase 22 builds this table from QRTN, QITBL, and OD2TBL tables. Phase 30 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

② Last entry is followed by a byte of zeros.

RCDTBL  
(TIB 11)

Purpose

Store each level-01 record-name in FD Section until input/output verb processing.

Entry Frequency

One entry for each 01-level entry in FD section.

2	1	Variable
Displacement of entry for file in FNTBL	c	Record-name in EBCDIC

Phases Involved

Phase 10 builds this table from 01-level statements. Phase 11 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 (i d k)	Displacement (d) of i d k, right-justified	Length of REDEFINES object

Entry Frequency  
One entry for each REDEFINES clause.

Phases Involved  
Phase 22 builds 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 SYMDMP if specified.

3	3
Dictionary pointer for renamed item	Dictionary pointer for renaming item

Entry Frequency  
One 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 a READ from library.

1	Variable	1	Variable	1
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 01 builds this table from COPY...REPLACING clause in source program.  
Phase 01 uses this table to replace data-names while source statements are being read from a library.

① Last entry followed by a byte of zeros.

RLDTBL (Phase 60)

- ① 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 defining item ②	Target address	Priority (0, if no segmentation)	Value of ADCON

Entry Frequency  
One entry for each PN definition, VN definition, and ADCON in Data A-text.

Phases Involved  
Phase 60 builds this table during Data A-text processing. Phases 60 uses this table to punch RLD and text cards for object module.

- ① There is no TIB for this table. Phase 60 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 60 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.

- ② Code    Meaning  
84    Data-A text RLD  
94    GN or PN information to be used to match VN definitions  
A4    VN definitions  
C4    VCONTBL ADDR INITI ADDR

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 defining item ①	Target address	Priority (0, if no segmentation)	Value of ADCON

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 punch RLD and text cards for the object module and VN EQUATE GN addresses.

- ① Code    Meaning  
10    RPT-ORIGIN GN  
84    DTF address  
94    PN definition  
A4    VN definition  
C4    INIT1 address

RNMTBL (Phase 22)

(TIB 12) Purpose  
Store information on objects of REDEFINES clauses.

1	3	1
Level number of REDEFINES subject	Pointer to dictionary entry for object	Dictionary minor code for object

Entry Frequency  
One entry for each REDEFINES clause.

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.

32	2	1	2	1
Data-name for report group in EBCDIC 1	GN number for this report-group	NOP code 00 = NOP PLUS 01 = NOP ZERO	Displacement of entry in RWRTEL for report-name associated with this data-name	1

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 11 uses this table to generate coding in response to USE BEFORE REPORTING statements.

- ① Left-justified, padded with binary zeros in low-order bytes.
- ② Code for unique report group names

Code	Meaning
00	This entry was made as the result of a unique report group name.
01	This entry is a non-unique report group name that must be qualified.

- ③ First entry is a dummy.

ROLTBL (TIB 15)

Purpose  
Store the SUM clause data-names and operand-names that are needed to create ROL-ROUT and RST-ROUT routines.

2	1	1	2	2	4
① Unused	SUM level	Unused	Displacement of SUM name entry in SNMTBL. This is the item to be rolled forward	Displacement of sum name in SNMTBL or nnnn portion of S.nnnn. This is the item summing takes place	②

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 routines.

4	4
Displacement of sum name in SNMTBL or nnnn portion of S.nnnn. This is the item into which the summing takes place	Contains zeros to indicate end of entry

① . Contains zeros if last entry in table.

Bytes	Contents
0	FF
1	00
2-3	Displacement

ROUTBL  
(TIB 16)

Purpose  
Store GNs for routines in each Report Writer generated subprogram.

Entry Frequency  
One entry for each report in Report Section.

Phases Involved  
Phase 12 builds and uses this table.  
Phase 11 uses this table in Report Writer verb processing

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	Unused

2	2	2	2	2
GN number for CHF-ROUT	GN number for CFF-ROUT	GN number for PH-1	GN number for LAST ROLL	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

RWRTBL  
(TIB 13)

Purpose  
Store information on a report-name.

Entry Frequency  
One entry for each report-name.

Phases Involved  
Phases 10 and 12 build this table from scan of FD and RD entries.  
Phases 12 and 11 use this table during scan of Report Writer verbs.

30	5	2	2
Report-name in EBCDIC	-nnnn portion of record-name for file-name-1	Pointer to entry in FNTBL	Size in binary of larger record

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

- ① Left-justified, padded with binary zeros in low-order bytes.
- ② Contains zeros if report is to be written on only one file.

SATBL  
(TIB 5)

Purpose  
Store file-names associated with SAME AREA clauses until all SAME AREA clauses have been processed.

1	2	1	Variable	1	Variable
Count of number of files in clause	Card Number	c	File-name in EBCDIC	c	File-name in EBCDIC

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.

SDSRATBL  
(TIB 5)

Purpose  
Determine the size of the SAME RECORD AREA and the record boundary within the area for SORT records.

1	1	1	2
SAME RECORD AREA number	BL number	Number of BLs	Maximum record size

Entry Frequency  
One entry for every SD sharing SAME RECORD AREA with an PD.

Phase Involved  
Phase 21 SORTPROC routine builds this table. SRA routine uses the table to build the SRAMAX table and also to generate SAME RECORD AREAS.

SEGINDX  
(TIB 16)

Purpose  
Store information about program fragments if SYMDMP or STATE is specified.

1	3	3	3
Priority	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 SYS002 and building the PROCTAB table. COBOL library subroutines use this table.

① For the SYMDMP option, the field contents are:

Bits	Contents
0-14	Relative block number in PROCTAB
15-23	Displacement within block

For the STATE option, the field contents are:

Bits	Contents
0-23	Displacement from the beginning of the PROCTAB entries in the object module

SEGTBL  
(TIB 15)

Purpose  
Store disk address of sections of Procedure A-text.

1	4
Priority number	Device address ①

Entry Frequency  
One entry for each segment control break.

Phases Involved  
Phase 51 creates an entry when it finds a segment control break. Gets priority from PNOU + 1 in phase 51, and device address from cell SEGSAV in phase 00. Phase 60 or, when OPT is specified, phases 62 and 63 use this table to combine sections into a segment.

Byte	Contents
0-1	Relative track number
2	Block number on track
3	Record identification

SETTBL  
(TIB 21)

Purpose  
Build SET strings for the SET statement after processing the final operand of the statement; or determine whether the ON SIZE ERROR option is present before building strings for ADD and SUBTRACT statements with multiple receiving fields.

1	Variable
c	P1-text element for operand

Entry Frequency  
One entry for each operand before the TO, UP BY, or DOWN BY options of the SET statement; or one entry for each operand between either the first receiving field and ON SIZE ERROR or the next verb after the ADD (or SUBTRACT) statement. The table is cleared at the end of processing for each verb.



Phases Involved

Phase 40 builds this table to store operands before the TO, UP BY, and DOWN BY options of the SET statement.

Phase 40 also uses this table to store the operands after the receiving field and before either the ON SIZE ERROR option or the verb following the ADD (or SUBTRACT) verb.

Phase 40 uses the table to build SET strings after processing the operand following the TO, UP BY, and DOWN BY key words of the SET statement.

Phase 40 uses the table to determine whether the ON SIZE ERROR option is present before issuing the string for the receiving fields after the first in an ADD or SUBTRACT statement.

SMSTBL  
(TIB 28)

Purpose  
Store SUM clause operand-names for correlation of SUM and SOURCE clauses.

2	c
c	SUM clause operand-name in EBCDIC

①

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 tables to generate a USM-ROUT routine for each detail report group and to build the ROLTBL table.

① First entry is a dummy.

SNMTBL  
(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 ROLTBL table, and generate USM-ROUT routines.

① Left-justified, padded with binary zeros in low-order bytes.

② First entry is a dummy.

SPNTBL  
(TIB 21)

Purpose  
Store function-name information from SPECIAL-NAMES paragraph in Environment Division.

3	1	Variable
①	c	Mnemonic-name in EBCDIC

Entry Frequency  
One entry for each function-name implementor.

Phases Involved  
Phases 10 builds this table from SPECIAL-NAMES paragraph.  
Phases 11 and 12 uses this table to substitute function-name for mnemonic-name in Procedure Division.

① Three possible configurations are:

- |    | <u>Byte 0</u>                 | <u>Byte 1</u>  | <u>Byte 2</u> |
|----|-------------------------------|--|---------------|
| a. | 1-character literal in EBCDIC | Unused   | Unused        |
| b. | 54                            | Code for device used (see COBOL word list under P0-text in "Section 5. Data Areas").                           | Unused        |
| c. | 55                            | COBOL word code for carriage control word (see special name element under P0-text in "Section 5. Data Areas"). | Unused        |

SRAMAX  
(TIB 10)

Purpose  
Determine the size of the SAME RECORD AREA and the record boundary within the area.

1	1	1
SAME RECORD AREA number	Maximum BL adjustment factor	Maximum record size

Entry Frequency  
One entry for every SAME RECORD AREA number.

Phase Involved  
Phase 21 builds this table on entry to routine SRA using the SRATBL, BLTABL, and SDSRATBL. Routine SRA uses this table to generate SAME RECORD ARFAs.

SRATBL (Phase 10)  
(TIB 6)

Purpose  
Store file-names associated with SAME RECORD AREA clauses until all SAME clauses have been processed.

1	2	1	Variable	1	Variable
Count of number of files in clause	Card number	c	File-name in EBCDIC	c	File-name in EBCDIC

Entry Frequency  
One entry for each 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.

SRATBL (Phase 21)  
(TIB 9)

Purpose

Store address of buffers for files with SAME RECORD AREA clauses.

1	1	2	1
DTF number	SAME RECORD AREA number	Size of area	BL number

Entry Frequency

One entry for each file named in a SAME RECORD AREA clause.

Phases Involved

Phase 21 builds by extracting files entered in the BUFTAB table with a SAME RECORD AREA clause. Phase 21 uses for buffer generation.

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	c	Name of key in EBCDIC
02 = DESCENDING		

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 22 uses this table to make sure keys named are defined in group. If not, sets error bit in INDKEY table for Phase 30 reference.

SRCTBL  
(TIB 22)

Purpose

Store SOURCE clause operand names to correlate SOURCE and SUM clauses.

2	2	Variable
Length of variable field	Displacement into DETTBL table of detail report group data-name	SOURCE operand with all qualifiers, indexes, and subscripts, if any, in P0-text format.

Entry Frequency

One 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 tables to generate a USM-ROUT routine for each detail report group.

SSATBL  
(TIB7)

Purpose

Store file-names associated with SAME SORT AREA clauses until all SAME clauses have been processed.

1	2	1	Variable	
Count of number of files in clause	Card number	c	File-name in EBCDIC	[Repeat once for each file in clause]

Entry Frequency

One entry for each SAME SORT AREA clause.

Phases Involved

Phase 10 builds from SAME SORT AREA clauses in the source program. Phase 10 uses to check SAME SORT AREA clause syntax.

STRING  
(TIB 9)

Purpose

Store verb strings while they are being built for output as P2-text.

Variable	1
Verb string	①

Entry Frequency

One entry for each operand in current string.

Phases Involved

Phase 40 builds as strings are processed. Phase 40 uses this table to collect output before generating.

① This field contains the number of bytes in the preceding field.

SUMTBL  
(TIB 19)

Purpose

Store data-names and operand-names from SUM clauses that are used to create routines USM-ROUT, INT-ROUT, RST-ROUT, and ROL-ROUT.

2	1	1	2	1
n	Unused	SUM level	Generated card number for this SUM clause	Reset level
①				

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 routines USM-ROUT, INT-ROUT, RST-ROUT, and ROL-ROUT.

2
Displacement of entry in DETTBL for detail name in SUM...UPON clause

1	4	7	24
Code for next field (2)	Pointer to SUM name in SNMTBL or nnnn portion of S.-name	E.-name (REDEFINES) in P0-text format (3)	PICTURE for name in EBCDIC

2	2	2
Displacement of entry in SMSTBL for first operand-name in SUM clause	Displacement of entry in SMSTBL for last operand-name in SUM clause	Zeros

(1) Contains zeros if last entry in table.

(2) Code      Meaning  
 00          2 bytes contain displacement into SNMTBL table  
 10          Next 4 bytes contain nnnn portion of S.-name  
 FF          Next field contains nnnn portion of S.-name

(3) Byte      Contents  
 0          06  
 1          E  
 2          . (period)  
 3-6        nnnn

TGTADTBL  
 (TIB 18)

Purpose  
 Gather information needed by phase 65 for processing the SYMDMP, STATE, and FLOW options.

4	4
Displacement of Debug table from beginning of TGT	ID number of last card written by phase 60

Entry Frequency  
 Information entered depends on options in effect.

Phases Involved  
 Phase 60 builds this table.  
 Phase 65 uses this table in processing SYMDMP, STATE, and FLOW options.

4	4
Relative address of byte following last byte of INIT3 (1)	Relative address of 0-routines, or if none, INIT2 (2)

4	4
Relative address of first instruction in Declarative Section (2) (3)	Relative address of START (2)

2	2
Displacement of first DTF cell from beginning of TGT ② ④	Displacement from beginning of PGT to virtual for ILBDTEF3 (0 if virtual not present) ② ④

- ① If FLOW is in effect, phase 65 changes this value to the relative address of the byte following the last byte allocated for the use the COBOL library subroutine for the FLOW option (ILBDPLW0).
- ② The field is not present if FLOW is the only option in effect.
- ③ The field contains the relative address of START if there are no declaratives.
- ④ The field is allocated but not filled in if the STATE option is in effect.

UPSTBL  
(TIB 25)

Purpose  
Store UPSI-switch byte information for phase 22 use in dictionary processing.

2	2
n	Card number of UPSI-x name

Entry Frequency  
One entry for each UPSI name.

Phases Involved  
Phases 10 builds from Special-names paragraph and Data Division. Phase 22 uses in dictionary processing.

1	6	9	1	Variable	9
	UPI-x	Dictionary	c	Mnemonic-name in EBCDIC	Dictionary attributes of mnemonic-name

1	Variable	10	1	Variable
c	Condition-name for ON or OFF STATUS in EBCDIC	Dictionary attributes of condition-name	c	Condition-name for OFF or ON STATUS in EBCDIC

10
Dictionary attributes of condition name

USNGTBL  
(TIB 2)

<u>Purpose</u> 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)

<u>Purpose</u> Save Data A-text address constant definitions object for group items containing VALUE clauses.	3	3	1	2	Variable
	X'100028'	Target address	Code ①	Size of constant in bytes	Value of constant

Entry Frequency  
One entry for each group item with a VALUE clause that is currently being processed.

Phases Involved  
Phase 20 builds this table from Data IC-text LD entries.  
Phase 22 uses this table to generate Data A-text entries.

①	<u>Code</u>	<u>Meaning</u>
	01	Alphanumeric literal
	FF	ALL or a figurative nonnumeric constant.

VALTRU  
(TIB 33)

<u>Purpose</u> Store literals for VALUE...THRU clause or VALUE clause in level 88 group item.	1	Variable	1	Variable	1
	c ①	P1-text literal element	c ②	P1-text literal element	FF ③
	④		④		③

Entry Frequency  
One entry for each value in each VALUE clause of this type.

Phases Involved  
Phase 20 builds from Data IC-text LD entries.  
Phase 30 uses to fill in P1-text literals with the actual values.

Phase 22 uses for syntax-checking of the VALUE IS SERIES clause.

- ① In the high-order bit:  
0 = value is not followed by THRU  
1 = value is followed by THRU
- ② This portion of the entry follows the format of a P1-text element as follows:
 

Type	Meaning
32	Numeric literal
33	Floating-point literal
34	Alphanumeric literal
39	ALL constant
- ③ Indicates the end of the entries for a VALUE clause.
- ④ If the count field is all zeros, the entry is a dummy entry for a group item whose subsequent Level-88 items are in error and were ignored by phase 20.

**VARLTFL**  
(TIB 15)

Purpose  
Store information about variable-length items needed for the DATATAB table if SYMDMP is specified.

3	3
Dictionary pointer for variable-length items	Maximum size (including 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 10)

Purpose  
Control GN numbers branched to in PERFORM...VARYING.

3	3	3	3
GN number for condition branch	VN number for varied branches	GN number for PLUSGN	GN number for MOVEGN

Entry Frequency  
One entry for each PERFORM...VARYING.

Phases Involved  
Phase 40 builds this table from PERFORM...VARYING strings in P1-text.  
Phase 40 uses this table to issue P2-text strings with correct branches for different steps.



VERBDEF  
(TIB 14)

Purpose

Store information about the occurrences of COBOL verbs.

0	1	1	1	1
48	Number of occurrences	E0	Alphabetic verb sequence number	00

Entry Frequency

One entry for each COBOL verb used.

1	1	Variable
Length	FB	Verb-text

Phases Involved

Phase 11 builds this table when VERBREF or VERBSUM is specified. Phase 22 uses this table to generate verb DEF-text.



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 60 or, when OPT is specified, phase 62 builds this table when it builds table CVIRTB.  
Phase 60 or, when OPT is specified, phase 64 uses this table with table CVIRTB to eliminate duplicate virtuals. After PGT allocation, each entry points to entry in PGT virtual field.

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 EQU PN or VN EQU GN element encountered during Optimization A-text processing.

Phases Involved

Phases 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

Code	Meaning
0C	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 60 or, when OPT is specified, phase 62 builds from VN DEF-text elements in Optimization A-text.  
Phase 60 or, when OPT is specified, phases 63 and 64 sort 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-ranges of PERFORM statements.

2	2
PN number	VN number corresponding to PN

Entry Frequency  
One entry for each procedure-name.

Phases Involved  
Phase 40 builds this table from P1-text PNs and VNCTR in COMMON.  
Phase 40 uses this table to modify addresses and set up return VNs.

XAVAL

(TIB 2)

Purpose  
Optimize use of arithmetic temporary storage by object module.

2
ID number of 8-byte slot available in temporary storage

Entry Frequency  
One entry for each 8-byte slot.

Phases Involved  
Phase 50 makes 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 40 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 temporary storage	Relative pointer in temporary storage

Phase 50 uses this table to process compile-time arithmetic verbs.

1	1	2
Register number	Characteristics of operand <sup>①</sup>	Intermediate result number

<sup>①</sup> 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

XSCRPT  
(TIB 3)

Purpose

Store subscript and index information for optimization.

2	2	1	3	4
n+2	Number of subscripts or indexes	0	New addressing parameter of subscripted or indexed item <sup>①</sup>	Dictionary pointer to unique identifier of subscripted item <sup>②</sup>

Entry Frequency

One entry for each subscripted or indexed item.

Phases Involved

Phase 50 builds this table from subscript verb string passed by phase 40.

Phase 50 uses this table to calculate address of subscripted or indexed item, or to generate object code for the calculation.

	1	3		1	3
	Flag	Dictionary pointer		Flag	Dictionary pointer
	byte	to unique identifier		byte	to unique identifier
	(3)	of first subscript		(3)	of third subscript
		or index-name (4)			or index-name (4)

- ① Bit      Meaning  
 0-3      3 = byte 2 contains number of register which at object time contains new address.  
 6 = bytes 2 and 3 contain the number of a SUBSCRIPT CELL which at object time contains the new address.

If bits 0-3 contain any other value, then the configuration is as follows:

<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

- ② Bits      Contents  
 0- 9      Zeros  
 10-22      Dictionary section number  
 23-31      Displacement in section

- ③ Bit      Meaning, if on  
 0      Literal  
 1-7      Unused

- ④ Bits      Contents  
 0- 1      Zeros  
 2-14      Dictionary section number  
 15-23      Displacement in section

XSSNT  
 (TIB 4)

Purpose

Store pointers to XSCRPT table during calculation of subscripted or indexed addresses.

2	2
ID number	Displacement in XSCRPT
of subscript	table of new address
or index	parameter of subscripted
computation	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.

TEXT FORMATS

This chapter contains diagrams of the formats of the texts used by the compiler. The diagrams are arranged in the following order:

1. Data Translation Texts: Data IC-, ATF-, and Data A-texts.
2. Procedure Translation Texts: P0-, P1-, P2-, Procedure A-, Optimization A-, and Procedure A1-texts.
3. E-text.
4. XREF-text.
5. Debug-text.

With some exceptions, one IC-text element represents one source element. (IC-text here refers to Data IC-, ATF-, P0-, P1-, or P2-text; 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 A-text the first two bits of this byte contain a code with the following significance:

Code	Meaning
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.

The following notes apply to the format diagrams in this chapter:

- The top row of figures shows the byte number for each field except where the preceding fields include a variable-length field.
- Broken lines indicate fields that are present only if the condition they satisfy is present.
- c = Number of bytes in the following field.
- n = Total number of bytes to follow in the text element.
- 1b = Length of field is one byte.
- Individual notes, applying to particular fields, are numbered consecutively with the numbers encircled.
- Double sets of characters in bytes 0 and 1 represent hexadecimal numbers.

DATA IC-TEXT

LD ELEMENT

0	1	2	3-4	5	6	7-8	9	10
03	n	Level indicator	Compiler-generated source card number	Switch byte	Switch byte	OCCURS DEPENDING ON maximum occurrences	Switch bytes	Number of indexes following
		①		②	③		④	

11	12	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	

**Note:** A series of logical records can follow the LD element. The types of records are ordered as follows:

Value for condition-name with multiple values	1b	1b	Variable
	Switch byte	c	Encoded VALUE
	(A)		

Indexes (first) or Keys (second)	1b	1b	Variable
	Flag	c	Index-name in EBCDIC
	(B)		

RENAMES or THRU name	1b	1b	Variable
	ID code	c	Name in EBCDIC
	(C)		

(A)	<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

(B)	<u>Bits</u>	<u>Meaning</u>
	0-3	Zeros
	4	Unused
	5	1 = INDEXED BY
	6	1 = DESCENDING KEY
	7	1 = ASCENDING KEY

(C)	<u>ID 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.



SD ELEMENT

0	1	2	3	4-5	6-7	8-9
03	n	Level indicator 36 (hex)	8	Compiler-generated source card number	Minimum RECORD CONTAINS value	Maximum RECORD CONTAINS value
			7			

10	11	12	13	14-43
9	SAME RECORD AREA number	Device code	c	Sort-name in EBCDIC, low order unused bytes padded with blanks
			10	

RD ELEMENT

0	1	2	3-4	5	Variable	1b
03	n	Level indicator 34 (hex)	Compiler-generated source card number	c	User-assigned EBCDIC report-name	00

FD ELEMENT

0	1	2	3-4	5-11	12	13	14	15	16	17-18
03	n	Level indicator 38 (hex)	Compiler-generated card number	11	SYS number in binary	Switch byte	Switch byte	Hopper switch	Buffer offset	Displacement of entry for file in PIOTBL
					12	13	13a			

19	20	21-22	23-24	25-26	27-28
Device and organization code for associated files	Switch byte	Integer-1 specified in BLOCK CONTAINS	Integer-1 specified in RECORD CONTAINS	Integer-2 specified in RECORD CONTAINS	Unused
	13b	14			

29-30	31-32	33	34	35	36	37	38	39
Unused	Displacement of entry for file in CKPTBL	SAME AREA number	Switch byte	Switch byte	Switch byte	SAME RECORD AREA number	Integer in POSITION option for this file	Number of CYL-OVERFLOW tracks
			15	16	17		18	

40	41	42	43-44	45	46	47-48	49	50	51-53	54	Variable
Device code	Device number of highest index	SAME SORT AREA number	Integer-2 specified in BLOCK CONTAINS	Switch byte	Reserved	Unused	(20a)	(20b)	Unused	c	File-name in FD entry
(10)	(19)			(20)							

Variable	Variable	Variable	Variable	Variable	Variable	Variable
TRACK AREA size	NOMINAL KEY and qualifiers	ACTUAL KEY and qualifiers	RECORD KEY and qualifiers	APPLY CORE-INDEX data-name and qualifiers	FILE STATUS data-name and qualifiers	PASSWORD data-name and qualifiers
(21)		(22)	(22)	(22)	(22)	(22)

Variable	Variable	Variable	1b
One byte of X'00' reserved	TOTALING AREA data-name and qualifiers	LABEL RECORDS names	00
	(22)	(23)	

- (1) Code (hex)      Meaning  
 01-31    = Levels 01-49  
 32      = Level 77  
 33      = Level 88  
 34      = RD  
 36      = SD  
 38      = FD  
         '-nnnn' name in Report Section  
 39      = Level 66

- (2) Bits      Code  
 0          1 = BLANK WHEN ZERO  
 1          1 = JUSTIFIED  
 2-4      Type of VALUE Clause  
 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      Type of USAGE  
 000 = No clause  
 001 = DISPLAY  
 010 = COMPUTATIONAL  
 011 = COMPUTATIONAL-1  
 100 = COMPUTATIONAL-2  
 101 = COMPUTATIONAL-3  
 110 = DISPLAY-ST  
 111 = INDEX

- (3) Bits      Code  
 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 (PICTURE field contains the E.nnnn from which PICTURE information is to be extracted)  
 6          1 = RENAMES data-name entry follows  
 7          1 = SYNCHRONIZED

- (4) Bits      Meaning, if SYNCHRONIZED  
 0          0 = SYNC LEFT  
           1 = SYNC RIGHT  
 1          RENAMES THRU data-name follows  
 2-4      000 = No SIGN clause  
           001 = TRAILING  
           011 = LEADING  
           101 = SEPARATE TRAILING  
           111 = SEPARATE LEADING  
 5-7      Unused

- (5) 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 with multiple values.

- ⑥ 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 8 bits of zeros.

⑦ Always contains 42 (hexadecimal).

Bits	Contents
0-3	Number of work units
4-5	Label Records are: 01 = Standard 10 = Omitted
6	1 = SAME RECORD AREA
7	1 = SAME SORT AREA

Bits	Contents
0-2	<u>Device Class</u> 000 Not specified 001 Direct Access 010 Unit Record 100 Utility
3-6	<u>Recording Mode</u> 1000 = F 0100 = V 0010 = U (invalid) 0001 = S
7	ASCII collating sequence

Code(decimal)	Device
1	1442R
2	1442P
3	2520R
4	2520P
5	2540R
6	2540P
7	2501
8	1403, 3203, or 5203
9	1404
10	1443
11	1445
12	3211
13	3505

14	3330
15	3525R
16	3525P
17	3525W or 3525M
18	Unused
19	3881
21	5425R
22	5425P
23	5425W
24	Unused
25	2560R
26	2560P
27	2560W
28-30	Unused
31	2311
32	Unused
33	2314 or 2319
34	2321
35	3340
36	3540
37-39	Unused
40	2400 or 3420 or 3410

⑪ Seven-byte external name or six-byte SYSnnn with a padding blank.

Bits	Code
0	1 = RANDOM ACCESS
1-3	Organization 000 Not specified or SEQUENTIAL 'S' 001 INDEXED 010 DIRECT with REWRITE 'U' or 'W' 011 DIRECT 'A' or 'D'
4-6	Device class 000 Not specified 001 DIRECT-ACCESS 010 UNIT-RECORD 100 UTILITY
7	1 = RESERVE NO ALTERNATE AREA

Bits	Meaning, if on
0	SELECT OPTIONAL
1	SAME AREA
2	EXTENDED SEARCH
3	SAME RECORD AREA
4	SAME SORT AREA
5	Pointer to CKPTBL in entry
6	Pointer to PIOTBL in entry
7	Word ALTERNATE specified in RESERVE clause

⑬a VSAM Support Byte Format

Bits	Meaning
0	1 = AS specified in ORGANIZATION parameter of system-name

1 1 = No organization parameter specified in system-name  
 2-4 ORGANIZATION clause

Code	Meaning
000	Not specified
001	SEQUENTIAL
010	INDEXED
100	RESERVED

5 ACCESS MODE IS DYNAMIC  
 6 Unused  
 7 PASSWORD data-name specified with RECORD KEY or for the file.

(15) Bits Code  
 0-1 TRACK AREA  
 00 = Not specified  
 10 = Integer  
 2 1 = Direct file with relative addressing  
 3 1 = NOMINAL KEY  
 4 1 = ACTUAL KEY  
 5 1 = RECORD KEY  
 6 1 = WRITE ONLY  
 7 FILE STATUS clause specified

(13b) Code Meaning  
 02 RCE (Read column eliminate) (3505 or 3525R)  
 04 Optical mark READ (3505)  
 08 PUNCH/PRINT (5424P, 3525P or 2560R)  
 80 READ/PUNCH/PRINT (5424R, 5424P, 5424W, 3525R, 3525P, 3525W, 2560R, 2560P, or 2560W)  
 90 READ/PUNCH/PRINT (3525M)  
 A0 READ/PUNCH (5424R, 5424P, 3525P, 3525R, 2560P, or 2560R)  
 C0 PUNCH/PRINT (5424P, 5424W, 3525P, 3525W, 2560P, or 2560W)  
 D0 PUNCH/PRINT (3525M)  
 E0 READ/PRINT (5424R, 5424W, 3525R, 3525W, 2560R, or 2560W)  
 F0 READ/PRINT (3525M)  
 82 READ (with RCE feature)/PUNCH/PRINT (3525R)  
 A2 READ (with RCE feature)/PUNCH (3525R)  
 E2 READ (with RCE feature)/PRINT (3525R)  
 01 RESERVE integer-1[ALTERNATE] AREA in binary (maximum is 1)

(16) Bits Meaning, if on  
 0 WRITE VERIFY  
 1 CYL OVERFLOW  
 2 'integer' of RESERVE clause not in valid range  
 3 Multiple REEL/UNIT  
 4 Multiple File Tape  
 5 MASTER-INDEX  
 6 CYL-INDEX  
 7 Unused

(17) Bits Meaning, if on  
 0-3 Number of SORT work units or number of reels with non-standard labels  
 4 CORE-INDEX  
 5 Unused  
 6 ASCII file  
 7 'integer' of ASSIGN not in valid range

(18) If not specified, file-names in multiple file clauses are assigned sequential numbers as encountered, starting with 1, and an entry is made in this field for each file.

(19) Initialized as 2311.

(14) Bits Code  
 0 1 = COPY  
 1 Unused  
 2 RECORD CONTAINS clause  
 3-4 BLOCK CONTAINS integer 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 = REPORTS clause

(20) Bits Meaning, if on  
 0-1 Unused  
 RECORDING MODE  
 2 Format F  
 3 Format V  
 4 Format U  
 5 Format S  
 6-7 Unused

(20a) Bit Meaning, if on  
 0 Incorrect class parameter of implementor

- 1 Incorrect device parameter of implementor name
  - 2 Incorrect organization parameter
  - 3-7 Unused
- ②0b Bit      Meaning, if on
- 0 Primary input hopper select
  - 1 Secondary input hopper select
  - 2-7 Unused

- ②1 2-byte field giving integer TRACK AREA count.

- ②2 Subfield                      Contents
- 1 2-byte count of bytes in all the subfields that follow in

- 2 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.

- ②3 Series of all label record-names preceded by 1-byte count of characters.

ATF-TEXT

Level	0	1	2	3-4	5-6	7-8	9	10	11-12
01-49 or 77 items	03	n	Level number	Generated card number	FLAG	Maximum number of occurrences	Number of indexes	Number of keys	Length of the item in the object module
			①			②			

13-Variable	Variable	Variable	Variable	Variable	Variable
EBCDIC name of item	EBCDIC name of object	Table displacement	Partial dictionary attributes	VALGRP table displacement	VALTRU table displacement
③	④	⑤	⑥	⑦	⑧

Level	0	1	2	3-4	Variable	Variable
88 items	03	n	(X'33' (Level number)	GENERATED card number	EBCDIC name of item	DICTIONARY attributes (partial)
					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, if on
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 ON2TBL 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. The field does not exist.

⑧ Either:

1. A 2-byte VALTRU table displacement if flag bit 6 is on: or
2. The field does not exist.

DATA A-TEXT

SECONDARY DTF ADDRESS

0	1-3	4
04	Relative address in object module of SDTF	Number assigned from SDTFCTR in COMMON

DTF ADDRESS

0	1-3	4
08	Relative address in object module of DTF	Number assigned from DTFCTR in COMMON

BLOCK ADDRESS

0	1-3	4	5-6	7-8
0C	Relative address in object module of buffer assigned to file.	BL number -- first base locator number assigned to file from BLCTR in COMMON	00	①

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	①

CONSTANT DEFINITION

0	1-3	4	5-6	7 through 6 + c
28	Relative address in object module where constant is located	Type ② of constant ③	c	Actual constant ③

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

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 TGT 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 TGT where displacement for base locator described in next field is to be placed	BLL number -- base locator number

DATA-NAME XREF ELEMENT

0	1-2	3-5	6	7 through 6+c
48	Card number in source program defining data-name	Pointer to dictionary entry of data-name	c	EBCDIC data-name

① Size, in words, of block section or area to which this entry refers.

②	Code (Hex)	Meaning
	00	Binary
	01	Alphanumeric
	FF	ALL constant

③ If the constant is an ALL constant, the format is different, beginning with byte 5, as follows:

5-6	7	8 through 7 + c
d	c	Value specified for the constant

where d is the number of bytes reserved for the constant.



PROCEDURE IC-TEXT (P0 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

EBCDIC data-name of GIVING option for USE error declarative

0	1	2 through 1 + c
24	c	User assigned EBCDIC name that was object of GIVING option

PN'S FOR ERROR/LABEL DECLARATIVES

0	1	2-3	4-5
26	0A	GN number for STANDARD ERROR	GN number for file header labels

6-7	8-9
GN number for file trailer labels	GN number for end-of-volume labels

10-11
GN number for begining-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 EXHIPIT NAMED statement

LISTING A-TEXT FOR PROCEDURE-NAMES

0	1	2 through 1 + c
37	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
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 (hex) = greater than
	0A (hex) = less than
	0C (hex) = not equal

PARENTHESIS

0	1
52	00 (hex) = left parenthesis
	01 (hex) = right parenthesis

ARITHMETIC OPERATOR

0	1
53	Operator code (4)

COBOL WORD

0	1	2
54	Word code (7)	Code (phases 10, 12, 11 only, not passed on) (8)

SPECIAL NAME

0	1
55	Code (5)

FIGURATIVE CONSTANT

0	1
75	EBCDIC value of figurative constant

STANDARD DATA-NAME REFERENCE

0	1
79	05 (hex) = TALLY

CARD NUMBER

0	1-2
81	Compiler-generated sequential source card number (9)

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

0	1
B9	(6)

(1) Each field contains zeros if there is GN for that purpose.

(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

		<u>Code</u>	<u>Meaning</u> <u>P0- and P1-text</u>	<u>P2-text</u>
0C	End of Report Writer Procedures			
0D	End of Segment			
0F	Date-Compiled entry	16		IF-EQ-NONNUM
F0	Security entry			
F1	Identification Division	17		IF-NOTEQ-NONNUM
F2	Program-ID entry			
F3	Author entry	18		IF-GT-NONNUM
F4	Environment Division			
F5	Configuration Division	19		IF-NOTGT-NONNUM
F6	Source-Computer entry			
F7	Object-Computer entry	1A		IF-LT-NONNUM
F8	Input-Output Section			
F9	File-Control entry	1B		IF-NOTLT-NONNUM
FA	I-O Control entry			
FB	Special-Names Section	1C	ALTER	MOVE-4
FC	Date-Written entry			
FD	Installation entry	1D	MOVE	MOVE
FE	Remarks entry			
		1E	EXAMINE	EXAMINE
		1F	TRANSFORM	TRANSFORM

③ Verb Code List: Code indicates the type of verb.

<u>Code</u>	<u>Meaning</u> <u>P0- and P1-text</u>	<u>P2-text</u>		
00	ADD	ADD	20	READ
01	SUBTRACT	SUBTRACT	21	OPEN
02	MULTIPLY	MULTIPLY	22	CLOSE
03	DIVIDE	DIVIDE	23	WRITE
04	COMPUTE	EXPONENTIATE	24	REWRITE
05		STORE	25	ACCEPT
06	END OF SENTENCE	IF-EQ-NUMERIC	26	DISPLAY
07	IF	IF-NOTEQ-NUMERIC	27	EXHIBIT
08	ELSE (OTHERWISE)	IF-GT-NUMERIC	28	RESET
09		IF-NOTGT-NUMERIC	29	READY
0A		IF-LT-NUMERIC	2A	RETURN
0B		IF-NOTLT-NUMERIC	2B	ON
0C		IF-ALPHABETIC	2C	ENTRY
0D		IF-NOT-ALPHABETIC	2D	CALL
0E		IF-NUMERIC	31	USE
0F		IF-NOT-NUMERIC	32	EXIT
10	STOP	STOP	33	REPORT-NOP
11	GO	GO	34	GENERATE
12		GO-DEPEND-FIRST	35	TERMINATE
13		GO-DEPEND-MIDDLE	36	SORT
14		GO-DEPEND-LAST	37	RELEASE
15		EVAL	38	PERFORM
			39	SUBSCRIPT
			3A	INITIATE

<u>Code</u>	<u>P0- and P1-text</u>	<u>Meaning</u>	<u>P2-text</u>	<u>Code</u>	<u>P0- and P1-text</u>	<u>Meaning</u>	<u>P2-text</u>
3B	DEBUG		DEBUG	5F	SEARCH ALL		End of WHEN in SEARCH ALL
3C			START (FORMAT-2)	60			SET format-1
3D			TRACE	61	SET		SET format-2 (UP BY)
3E			EQUATE	63	SEEK		SEEK
3F			MOVE-1	64	START		START
40	INIT		INIT	66			IF EQUAL (index name)
41	INCRA		INCRA	67			IF NOT EQUAL (index name)
42	STEP		STEP	68			IF GREATER (index name)
43	UPDATE		UPDATE	69			IF NOT GREATER (index name)
44			USE-ERROR	6A			IF LESS (index name)
45			ENDUSE-ERROR	6B			IF NOT LESS (index name)
46			USE-LABELS	6C	Virtual Definition		Virtual Definition
47			ENDUSE-LABELS	6D	EQUATE in SEARCH ALL		EQUATE in SEARCH ALL
4A			USE-REPORT	6E			SET format-2 (DOWN BY)
4B			ENDUSE-REPORT	6F			GO TO (Segmentation)
4C	Q-CALL		Q-CALL	70			Segmentation initialize
4D	Q-RETURN2		Q-RETURN2	73	GOBACK		GOBACK
4E	Q-RETURN3		Q-RETURN3	74			EXIT program
4F	REPORT-CALL		REPORT-CALL	76	SETVLC (for RENAMES Q-routine)		SETVLC (for Q-routine)
50	REPORT-SAVE-0		REPORT-SAVE-0	77			FLOW
51	REPORT-SAVE-1		REPORT-SAVE-1	79			OPEN (VSAM)
52	REPORT-SAVE-2		REPORT-SAVE-2	7A			CLOSE (VSAM)
53	REPORT-SAVE-3		REPORT-SAVE-3	7C	GNRPT (for OPT)		GNRPT
54	REPORT-SAVE-4		REPORT-SAVE-4	7E			READ (VSAM)
55	REPORT-SAVE-5		REPORT-SAVE-5	7F			WRITE (VSAM)
56	REPORT-RETURN-0		REPORT-RETURN-0				
57	REPORT-RETURN-1		REPORT-RETURN-1				
58	REPORT-RETURN-2		REPORT-RETURN-2				
59	REPORT-RETURN-3		REPORT-RETURN-3				
5A	REPORT-RETURN-4		REPORT-RETURN-4				
5B	REPORT-RETURN-5		REPORT-RETURN-5				
5C	REPORT-ORIGIN		REPORT-ORIGIN				
5D	REPORT-REORIGIN		REPORT-REORIGIN				
5E	SEARCH		Beginning of WHEN in SEARCH ALL				

<u>Code</u>	<u>P0- and P1-text</u>	<u>Meaning</u> <u>P2-text</u>	<u>Code</u>	<u>P0- and P1-text</u>	<u>Meaning</u> <u>P2-text</u>
80		REWRITE (VSAM)	<u>Code</u> 01	<u>Word</u> DATA	
81		START (VSAM)	02	SKIP1	
82	DELETE (VSAM)		03	SKIP2	
86	SERVICE	SERVICE	04	SKIP3	
87	MERGE		05	EJECT	
88		COUNT	06	NSTD-REELS	

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

⑥

<u>Error Symbol</u>	
COBOL word code	-- If reserved word used invalidly (See also note 7)
00 (hex)	-- If undefined or multiply-defined symbol found

⑦ COBOL Word Code:  
This list shows the code number assigned to each COBOL reserved word for use in Procedure IC-text (P0, P1, and P2). In the COBOL word table, COBWRD, in phases 10, 11, and 12 of the compiler listing, the words appear in alphabetical order according to their length.

07	SUPPFESS
09	SORT-OPTION
0A	ORGANIZATION
0B	WHEN-COMPILED
0C	CORE-INDEX
0D	PROGRAM
0E	RF
0F	WRITE-ONLY
12	COMMA
13	DECIMAL-POINT
14	FILE-LIMIT(S)
15	MODE
16	RECORDING
17	REEL
18	SYSIPT
19	SYSLST
1A	TRACK-AREA
26	DISPLAY
28	RESET
2E	ON
30	CURRENCY
32	INDEX
33	STATUS
34	MODULES
35	MEMORY
36	WORDS
37	SYNCHRONIZED (SYNC)
38	OFF

<u>Code</u>	<u>Meaning</u> <u>P0- and P1-text</u>	<u>P2-text</u>	<u>Code</u>	<u>Meaning</u> <u>P0- and P1-text</u>	<u>P2-text</u>
39	RENAMES		5A	END-OF-PAGE (EOP)	
3A	UP		5B	CHARACTER	
3B	DOWN		5C	NOT	
3C	FILE (in Procedure Division and after File Section header)		5D	AND	
3D	OPTIONAL		5E	OR	
3E	REMAINDER		5F	LIMIT(S)	
3F	POSITION		61	BEGINNING	
40	TAPE		62	ENDING	
41	TRAILING		63	MORE-LABELS	
42	ADDRESS		64	OUTPUT	
43	ALPHANUMERIC		66	INPUT	
44	NUMBER		67	RANDOM	
45	CURRENT-DATE		68	PROCESSING	
46	TIME-OF-DAY		69	BEFORE	
47	COM-REG		6A	REPORTING	
48	SORT-RETURN		6B	I-O	
49	SEPARATE		6C	WITH	
4B	REREAD		6D	REWIND	
4C	DISP		6E	REVERSED	
4D	EXTENDED-SEARCH		6F	INTO	
4E	MASTER-INDEX		70	AT	
4F	CYL-OVERFLOW		71	INVALID	
50	THEN		72	AFTER	
51	CYL-INDEX		73	ADVANCING	
52	WRITE-VERIFY		76	LOCK	
53	THAN		77	SYSPCH	
54	RECORD-OVERFLOW		78	CONSOLE	
55	ALPHABETIC		79	ALL	
56	NUMERIC		7A	CORRESPONDING (CORR)	
57	POSITIVE		7B	TALLYING	
58	NEGATIVE		7C	LEADING	
59	UPDATE		7D	UNTIL	
			7E	REPLACING	

<u>Code</u>	<u>Meaning</u> <u>P0- and P1-text</u>	<u>P2-text</u>	<u>Code</u>	<u>Meaning</u> <u>P0- and P1-text</u>	<u>P2-text</u>
7F	BY		A2	UNIT (S)	
81	GIVING		A3	FOR	
82	ROUNDED		A4	IN, OF	
83	SIZE		A5	SECTION	
84	ERROR		A6	LABEL-RETURN	
85	RUN		A7	DIVISION	
86	PROCEED		A8	SORT-FILE-SIZE	
87	THROUGH (THRU)		A9	SORT-CORF-SIZE	
88	VARYING		AA	SORT-MODE-SIZE	
89	USING		AB	SIGN	
8A	COBOL		AC	SORT (appears in Procedure Division as verb with 36 code)	
8B	UPSI-1 through UPSI-7		AD	MULTIPLE	
8C	DESCENDING		AF	FILLER	
8D	ASCENDING		B1	ASSIGN	
8E	TRACE		B2	ACCESS	
8F	CHANGED		B3	EXCEPTION	
90	NAMED		B4	RESERVE	
92	CHARACTER (S)		B5	NOMINAL	
93	TIMES		B6	ACTUAL	
94	DEPENDING		B8	DYNAMIC	
95	LINE (S)		BA	SEQUENTIAL	
96	FIRST		BC	INDEXED	
97	NEXT		BE	ALTERNATE	
98	UPON		BF	AREA (S)	
99	PROCEDURE		C1	RELOAD	
9A	EVERY		C4	TRACK (S)	
9B	TO		C6	CYCLES	
9C	IS, ARE		C8	PASSWORD	
9D	FROM		C9	EXTEND	
9E	NO		CA	VALUE (S)	
9F	KEY		CB	PRINT-SWITCH	
A0	RETURN-CODE		CC	BLOCK	
A1	END		CD	RECORD (S)	

<u>Code</u>	<u>Meaning</u> <u>P0- and P1-text</u>	<u>P2-text</u>
CE	CLOCK UNITS	
CF	RECORDS	
D0	CONTROL (S)	
D1	LABEL (S)	
D3	CONTAINS	
D4	OMITTED	
D5	STANDARD	
D6	REPORT (S)	
D7	REDEFINES	
D8	PICTURE (PIC)	
D9	BLANK	
DA	OCCURS,	
DB	JUSTIFIED (JUST)	
DC	POSITIONING	
DD	USAGE	
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	

<u>Code</u>	<u>Meaning</u> <u>P0- and P1-text</u>	<u>P2-text</u>
E7	FINAL	
E9	HEADING	
EA	DETAIL (DE)	
EB	LAST	
EC	FOOTING	
EE	GROUP	
EF	TYPE	
F0	PLUS	
F2	DISPLAY-ST	
F3	RH	
F4	PH	
F6	CH	
F8	CF	
FA	PF	
FB	SENTENCE	
FC	COLUMN	
FD	INDICATE	
FE	SOURCE	
FF	SUM	

⑧ Bits Meaning

0	FD, SD, RD
1	Paragraph word
2	Section word
3	Division word
4	Allowed in Environment Division
5	Allowed in Data Division
6	Allowed in Procedure Division
7	Allowed in Identification Division

⑨ The first bit of byte 1 is used as a flag. A setting of 0 indicates a PN statement. A setting of 1 indicates a verb statement.



PROCEDURE IC-TEXT (P1 FORMAT)

PROCEDURE-NAME DEFINITION

0	1	2 through 1 + c
06	c	Dictionary attributes of procedure name. See "Dictionary Entry Formats." ①

C+1
Priority
②

n - 7 to n - 6	n - 5 to n - 4
PN number for file trailer label	PN number for end-of-volume label

n - 3 to n - 2	n - 1 to n + 1
PN number for beginning-of-volume label	Pointer to dictionary entry for file ④

PROCEDURE-NAME REFERENCE

0	1	2 through 1 + c
20	c	Dictionary attributes of procedure name. See "Dictionary Entry Formats." ①

C+1
Priority
②

SD ELEMENT

0	1	2-9	10	11-12	n-1 to n+1
21	n	Dictionary attributes for SD (see "SD ENTRY")	0	GN number for Q routines	Dictionary pointer
			②A		

②A Bytes 10-12 are present only if the Q-bit is on.

FILE-NAME REFERENCE

0	1	2 through 21
21	n	Dictionary attributes of file. See "FD Entry" in Appendix F. ①

22	23-24
Count of number of Q-routine calls ③	GN number for Q-routines ③

n - 11 to n - 10	n - 9 to n - 8
PN number for STANDARD ERROR declarative	PN number for file header label

VSAM FILE-NAME REFERENCE

0	1	2 through 9	10
26	n	Dictionary attributes of file (see "FD ENTRY" in Appendix D)	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-routines	N number for STANDARD ERROR declarative ③	Reserved ③	Pointer to dictionary entry for file

DATA-NAME REFERENCE

0	1	2 to n - 5 or n - 2
30	n	Dictionary attributes. See "LD Entry" in "Section 5. Data Areas." (5)

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

n - 1 to n + 1
Pointer to dictionary entry for item (4)

DATA-NAME REFERENCE FOR KEY CLAUSE

0	1	2 to n - 6 or n - 3	n - 5
30	n	Dictionary attributes (see "LD ENTRY" in "Section 5. Data Areas") (1)	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 ACB number	Pointer to dictionary entry for item (4)

NUMERIC LITERAL

0	1	2	3
32	n	Positions to left of decimal	Positions to right of decimal

4 through n + 1
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	(6)

7-9
Pointer to dictionary entry for index-name (4)

LISTING A-TEXT FOR PROCEDURE-NAMES

0	1	2 through 1+c
37	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
37	n	EBCDIC verb	Alphabetic verb sequence number

DATANAME 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 (7)

VERB

0	1
44	Verb code (see note 3 under P0-text)

RELATIONAL CODE

0	1
50	06 (hex) = Equal
	08 (hex) = Greater than
	0A (hex) = Less than

PARENTHESIS

0	1
52	00 (hex) = Left parenthesis
	01 (hex) = Right parenthesis

ARITHMETIC OPERATOR

0	1
53	Operator code (8)

0	1
54	Word code (see note 7 under P0-text)

NFILES

0	1
56	Number of files in USING

FIGURATIVE CONSTANT

0	1
75	EBCDIC value of figurative constant

STANDARD DATA-NAME REFERENCE

0	1
79	05 (hex) = TALLY

CARD NUMBER

0	1-2
81	Compiler-generated sequential source card number (10)

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

0	1
B9	(9)

- ① Dictionary attributes without count and major code fields.

For alphanumeric edited items, elementary items with report pictures, and elementary items with sterling report pictures, phase 30 discards bits 10-17 while copying the dictionary attributes.

- ② Priority appended to procedure-name reference and definition by phase 3. Priority is part of dictionary attributes for section-names.

- ③ Bytes 22-24 are present only if the Q-bit is on.

- ④ Pointer contents:
 

Bits	Contents
0-1	Unused
2-14	Dictionary section number
15-23	Displacement in section

- ⑤ Dictionary attributes with flag byte field removed. Bits 1-4 of flag byte overlay bits 1-4 of level number. Bits 5-8 of flag byte overlay count field preceding major code field. In 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 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	UPSI-0
to	to
FF0007	UPSI-7
FF0008	CURRENT-DATE
FF0009	TIME-OF-DAY
FF000A	COM-REG
FF000B	SORT-RETURN
FF000C	SORT-CORE-SIZE
FF000D	SORT-FILE-SIZE
FF000E	SORT-MODE-SIZE
FF0010	NSTD-REELS
FF0015	WHEN-COMPILED

- ⑥ 

Bits	Contents
0	If 1, subject has variable length; bytes 5-6 contain VLC number. If 0, bytes 5-6 contain fixed length of subject.
1-3	Unused
4-7	1111

- ⑦ 

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
0F	Date-Compiled Entry
F0	Security Entry
F1	Identification Division
F2	Program-ID Entry
F3	Author Entry
F4	Environment Division
F5	Configuration Division
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 Section
FE	Remarks Entry

⑧

Code (hex)	Operator
00	Addition
01	Subtraction
02	Multiplication
03	Division
04	Exponentiation

⑩ The first bit of byte 1 is used as a flag. A setting of 0 indicates a PN statement. A setting of 1 indicates a verb statement.

⑨

<u>Error symbol</u>	
COBOL word number	-- If reserved word used invalidly (see also note 7 under "Procedure IC-text (PO Format)")
00 (hex)	-- If undefined or multiply-defined symbol found

PROCEDURE IC-TEXT (P2 FORMAT)

FILE-NAME REFERENCE

0	1	2	3	4	5	6	7	8-9
21	n	I/O verb options	Switches	First BL number	DTF pointer number	Switches	File number on multiple file reel	Maximum record length
		(1)	(2)			(3)		

10	11	12-13	14	15	16
Switches	Switches	User maximum label length	Access method	Print control communication	Unused
(4)	(5)		(6)	(7)	

17	18	19	20-21	22-24	25-27	n-14	n-13 to n-12
Secondary DTF pointer number	Secondary DTF pointer number	Secondary DTF pointer number	Block size for file	idk for ACTUAL KEY data-name if DA	Pointer to dictionary entry for ACTUAL KEY data-name if DA	0	GN numbers for Q-routines
				(9)		(8)	(8)
					(9)		

n-11 to n-10	n-9 to n-8	n-7 to n-6	n-5 to n-4
PN number for standard error declarative	PN number for file header label (BOF)	PN number for file trailer label (EOF)	PN number for end of volume label (EOV)

n-3 to n-2	n-1 to n+1
PN number for beginning of volume label (BOV)	Pointer to dictionary entry for file (10)

VERB INFORMATION

0	1	2 through 1 + c
24	c	Follows verb string for EXAMINE, TRANSFORM, EVAL, ADD, SUBTRACT, MULTIPLY, DIVIDE, USE, DEBUG.

VERB INFORMATION (VSAM) For VSAM READ, WRITE, REWRITE, DELETE, and START

0	1	2	3	4	5-7
24	c	ACB number	Execution-time information	Compile-time information	Dictionary pointer to RECORD KEY data-name
	(3e)	(3f)	(3b)	(3c)	(3d)

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-routines GNS	GN number for STANDARD ERROR declarative	Reserved	Pointer to dictionary entry for file

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 - 5	n - 4 to n - 3
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

n - 2	n - 1 to n + 1
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 "Section 5. Data Areas."	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	(13)	Index-name number	Length of subject (14)	Pointer to dictionary entry for item (10)	(13)

LISTING A-TEXT FOR PROCEDURE-NAMES

0	1	2 through 1 + c
37	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
37	n	EBCDIC verb	Alphabetic verb sequence number

MULTIPLE GN REFERENCE

0	1	2-3	n through n + 1
38	n	GN number	GN number

FIGURATIVE CONSTANT "ALL" (Greater than 1 character)

0	1	Variable
39	c	Alphanumeric literal following ALL

CRITICAL PROGRAM BREAK

0	1
42	Break code (15)



RELATIONAL CODE

0	1
50	06 (hex) = Equal
	08 (hex) = Greater than
	0A (hex) = Less than
	0C (hex) = Not less than

PHASE 40 OPTIMIZATION INFORMATION

0	1
43	Type code

(16)

COBOL WORD

0	1
54	Word code (See note 7 under "Procedure IC Text (PO Format)")

FIGURATIVE CONSTANT

0	1
75	EBCDIC value of figurative constant

STANDARD NAME REFERENCE

0	1
79	05 (hex) = TALLY

CARD NUMBER

0	1-2
81	Compiler-generated sequential source card number (21)

CARD NUMBER FOR FLOW

0	1-2
82	Card number

VERB

0	1	2
84	Verb code (See note 3 under "Procedure IC-text (P0 Format)")	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

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 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 (17)	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 (18)(19)	

08 CLOSE REEL  
09 CLOSE REEL, NO REWIND

- ① Bits 0-3: Not used
- Bits 4-7: Verb options, per table:

VERB	CODE	OPTION
OPEN	00	OPEN, INPUT
	01	OPEN, NO REWIND
	02	OPEN, REVERSED
	04	OPEN, OUTPUT
	0C	OPEN, INPUT/OUTPUT
READ	08	READ INTO
WRITE	08	WRITE FROM
CLOSE	00	CLOSE, REWIND
	01	CLOSE, NO REWIND
	02	CLOSE WITH LOCK

Bit	Meaning	Mask
0	If 1, Q-routines required	X'80'
1-2	As follows:	
	00 = Variable records	X'00'
	01 = Fixed records	X'20'
	10 = Undefined records	X'40'
	11 = Spanned records	X'60'
3	If 1, Multiple reel file	X'10'
4-7	Number of BLS for file	

- ③ Bits 0-3: Access method, as follows:

Bits	Access Method
0001	DTFCD
0010	DTFPR
0011	DTFMT
0100	DTFSD
0101	DTFDA
0110	DTFIS

Bit 4: Set to 1 if any of the following conditions occurs:

- OMR, if access method is DTFCDD.
- APPLY WRITE ONLY, if access method is DTFMT or DTFSD.
- ACTUAL KEY SPECIFIED, if access method is DTFDA for DA Sequential File.
- IBM extension (REWRITE) for DTFDA, Random File.
- Load type DTF generated, if access method is DTFIS.

Bit 5

- Set to 1 if access method is DTFDA and relative track addressing is used.
- Set to 1 if access method is DTFMT and file is an ASCII file.
- Set to 1 if access method is DTFIS and file is double buffered.

Bits 6-7: Not used.

(3b)	<u>Bit</u>	<u>Meaning</u>
	0	0 = SEQUENTIAL ACCESS or READ NEXT with DYNAMIC ACCESS
	1-7	1 = RANDOM or DYNAMIC ACCESS Unused

(3c)	<u>Bit</u>	<u>Meaning</u>
	0	0 = No duplicate string follows 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.

(4)	<u>Bit</u>	<u>Meaning, if ON</u>
	0	Blocked
	1-3	Not used
	4	SAME RECORD AREA
	5	Random access
	6-7	Label type, as follows:

- 00 - Standard
- 01 - User standard
- 10 - Nonstandard
- 11 - Omitted

(5)	<u>Code (Hex)</u>	<u>Meaning</u>
	80	USING/GIVING
	40	ACTUAL KEY in working storage
	20	BEFORE ADVANCING used in program*

- 10 AFTER ADVANCING used in program\*
- 08 AFTER POSITIONING used in program\*
- 04 Minimum case (single buffered, unblocked file or ISAM)
- 02 CLOSE with lock
- 01 OPEN optional

\*carriage control DTFPR

(6)	<u>Code (Hex)</u>	<u>Access Method</u>
	01	DTFCDD
	02	DTFPR
	03	DTFMT
	04	DTFSD
	05	DTFDA
	06	DTFIS

(7) This byte is used for print control communication between phases 40 and 51.

<u>Bit</u>	<u>Meaning</u>
0-1	Not used
2-3	As follows:
	00 = BEFORE ADVANCING
	01 = AFTER POSITIONING
	10 = AFTER ADVANCING
4-5	As follows:
	00 = Integer
	01 = Identifier (data-name)
	10 = Mnemonic Name
6	If 1, END OF PAGE test is required
7	Not used.

(8) Bytes n-14 to n-12 are present only if the Q-bit is ON.

(9) Bytes 22-27 are present only for a file-name reference in a READ or WRITE statement.

(10)	<u>Pointer Contents:</u>	
	<u>Bits</u>	<u>Contents</u>
	0,1	Unused
	2-14	Dictionary section number
	15-23	Displacement in section

(10c) Field contains zeros if GN is not generated.

(11) Dictionary attributes with flag byte field removed. Bits 1-4 of flag byte overlay bits 1-4 of level number. Bits 5-8 of flag byte overlay count field preceding major code field. In 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 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	UPSI-0
	to
FF0007	UPSI-7
FF0008	CURRENT-DATE
FF0009	TIME-OF-DAY
FF000A	COM-REG
FF000B	SORT-RETURN
FF000C	SORT-CORE-SIZE
FF000D	SORT-FILE-SIZE
FF000E	SORT-MODE-SIZE
FF0010	NSTD-REELS
FF0015	WHEN-COMPILED

If this data-name reference contains a subscript or index address calculation ID number, bit 0 will be on, bits 1-7 will contain 0, and bits 8-23 will contain the ID number. The high-order bit is turned on by Phase 40 when it assigns the ID number.

- 12) Note: Addressing Parameters field in attributes in "ID entry" has been replaced by unique subscript identifier element to match entry in DEFSBS table.

For alphanumeric edited items, elementary items with report pictures, and elementary items with sterling report pictures, Phase 30 discards bits 10-17 while copying the dictionary attributes.

- 13) This field is present only if the indexing is relative. When present, it contains a 2-byte binary literal, the object of the plus or minus.

14) Bit      Contents

0	1 = Subject has variable length: bytes 5-6 contain VLC number
	0 = Bytes 5-6 contain fixed length of subject
1-3	Unused
4-7	1111

15) 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
0F	Date-Compiled entry
F0	Security entry
F1	Identification Division
F2	Program-ID entry
F3	Author entry
F4	Environment Division
F5	Configuration Division
F6	Source-Computer entry
F7	Object-Computer entry
F8	Input-Output Section
F9	File-Control entry
FA	I-O Control
FB	Special-Names Section
FC	Date-Written entry
FD	Installation entry
FE	Remarks entry

16) Code (hex)      Meaning

02	Precedes a Procedure-name definition element equated to a VN for an ALTER verb, each generated Procedure-name defined for INVALID KEY/AT END, each procedure-name definition following TO PROCEED TO in an ALTER statement, each USE declarative, and each ENTRY verb. PERFORM...TIMES
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 any performed procedure except a PERFORM...TIMES procedure in a non-segmented program.

17) Code (Hex)      Meaning

02	SAVF-AREA
04	SWITCH
06	Unused
08	DEBUG
0A	Unused

18) Code (Hex)      Meaning

00	SDTFADR
04	VLC
08	CNCTL
0C	PFMCTL
10	PFMSAV
14	DTFADR
18	XSA
1C	PARAM
1D	Single-precision

floating-point  
 1E Double-precision  
 floating point  
 20 WORKING CELLS  
 24 TEMPORARY STORAGE  
 28 XSASW  
 2C BL, SPL, BLL  
 30 VIRTUAL  
 34 FIB

When the code for the preceding field is 1D or 1E, this field contains zeros.

19 Except when the code for the preceding field is 2C, 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:

20 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)

<u>Bits</u>	<u>Field</u>	<u>Value</u>	<u>Meaning</u>
0-3	i	0000	BL
		0001	BLL
		0100	SBL
4-7	-	0000	Unused
8-15	k		Base locator number assigned from corresponding COMMON counter.

21 The first bit of byte 1 is used as a flag. A setting of 0 indicates a PN statement. A setting of 1 indicates a verb statement.

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	Sequentially generated card number <sup>①</sup>

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	GN number -- number assigned from GNCTR in COMMON

VARIABLE PROCEDURE-NAME DEFINITION

0	1	2-3
38	<sup>②</sup>	VN number -- number assigned from VNCTR in COMMON

EBCDIC PROCEDURE-NAME GENERATOR

0	Consists only of this one-byte 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 <sup>③</sup>	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

PROCEDURE-NAME REFERENCE

0	1	2-3
4C	<sup>②</sup>	PN number assigned at point of definition from PNCTR (COMMON)

GENERATED PROCEDURE-NAME REFERENCE

0	1-2
50	GN number assigned at point of definition from GNCTR (COMMON)

VARIABLE PROCEDURE-NAME REFERENCE

0	1	2-3
54	<sup>②</sup>	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

GLOBAL TABLE STANDARD AREA REFERENCE

0	1	2
60	Type code (5)	Displacement in bytes from start of specified area

ADDRESS INCREMENTS

0	1-3
80	Value that is to modify an address. Negative value in 2's complement.

GLOBAL TABLE OTHER 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 60 ELEMENTS

(10)

BASE AND DISPLACEMENT

0	1-2	
70	Bits	Contents
	0-3	Register number
	4-15	Displacement

REGISTER SPECIFICATION

0	1
A0	Register number: 00 through 0F (hex)



INCREMENTED ADDRESS

0	1-3	4-6
A4	Addressing parameter (7)	Dictionary pointer (8)

7-9
Value of increment

SPECIAL PHASE 60 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 sub-scripted data-name

FILE REFERENCE ELEMENT

0	1-3
B8	Pointer to dictionary entry for file

PARAMETER FOR CALL TO SEGMENTATION AND GO TO DEPENDING ON SUBROUTINES

0	1	2-3
BC	Priority	PN number

PHASE 5 OPTIMIZATION INFORMATION

0	1
C0	Type code (13)

RPT-ORIGIN

0	1-2
D4	GN number

- (1) The first bit of byte 1 is used as a flag. A setting of 0 indicates a PN statement. A setting of 1 indicates a verb statement.
- (2) Byte 1 contains the priority number of the 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 Declarative Section; produced only if SYMDMP or STATE 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.
28	RESERVE -- specifies registers not to be used by phase 60 or phases 62 and 63 followed by register specification element.
2C	FREE -- indicates registers no longer reserved; followed by register specification element.

30 DESTROY -- indicates that contents of register 14 or 15 were destroyed; followed by register specification element. (phase 60 or Phase 63 generates a reload for the register from the appropriate cell when it needs the address contained there.)

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.

7	Bits	Field	Code	Content or Meaning
	0-3	i	0000	BL
			0001	BLL
			0011	Address of data-name as in register specified by bits 12-15
			0100	SBL
			0110	Subscript cell
	4-15	d		Displacement from start of area controlled by base locator.
	16-23	k		SBL, BL, or BLL number assigned from SBLCTR, BLCTR, or BLLCTR in COMMON or subscript cell number.

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	Checkpoint counter
	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)

4	Bits	Code	Meaning
	0-3	0000	Unused
	4-7	0000	BL
		0001	BLL
		0100	SBL

5	Code (hex)	Meaning
	02	SAVE-AREA
	04	SWITCH
	06	TALLY

6	Code (hex)	Meaning
	00	SDTFADR
	04	VLC
	08	ONCTL
	0C	PFMCTL
	10	PFMSAV
	14	DTFADR
	18	XSA
	1C	PARAM
	20	Working Cells
	24	Temporary Storage
	28	XSASW
	2C	SUBADR
	30	Temporary Storage-2
	34	FIB
	3C	SAVE-AREA-2
	44	REPORT-SAVE area
	48	Global Table Overflow cell
	50	Checkpoint Counter
	54	Temporary Storage-3
	58	Temporary Storage-4
	5C	Index-name
	60	IOPTRCTR

- ⑩ The following special Phase 60 A-text elements are generated and then used by phase 60 to generate MVC instructions to initialize VN cells in the TGT:

Identifier	Text Element
<u>Byte</u>	
90	A 4-byte element that contains the number of 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

<u>Code (hex)</u>	<u>Meaning</u>
04	TALLY
08	PARAM
0C	BL
10	SAVE2
14	SEL
18	FIB
1C	VLC

20	PN
24	GN
28	BLL
2C	Literal
60	SUBADR
64	Temporary Storage

<u>Code (hex)</u>	<u>Meaning</u>
00	No preceding base code
01	Base code precedes this element

<u>Code (hex)</u>	<u>Meaning</u>
01	Precedes a load instruction which is not followed by a branch instruction
02	Precedes any possible entry point for which addressability must be established
03	Precedes a load instruction for a PN or GN which must be processed with an address constant cell in the PGT
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 any performed procedure except a PERFORM...TIMES procedure in a nonsegmented program
06	Precedes a load instruction for a PN or GN which is followed by an unconditional branch instruction

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 code	c	Value of literal
	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) to identify a location

[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 code	c	Value of literal
	①		

SOURCE PROCEDURE-NAME EQUATE STRING (NON-OPTIMIZER VERSION)

0	1	2-3
0C	n / 2	First PN number assigned (Number of fields to follow) to identify a location

[Variable number of 2-byte fields containing GN numbers]

n through n + 1  
Last GN number assigned to same location as others in string

SEGMENTATION ELEMENT

0	1	2-3
14	Priority number of segment to which VN belongs	VN number

VIRTUAL CONSTANT ELEMENT

0	1-2	3-10
18	Contents of LOCCTR in COMMON	External-name of routine to be called

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 verbs 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 FOR 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 assigned 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 Bit	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.

<u>Code</u>	<u>Element Name</u>
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 OTHER 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
EC	SEGMENTATION AND GO TO DEPENDING ON SUBROUTINE CALL PARAMETER

The SPECIAL PHASE 60 ELEMENTS are also present in Procedure A1-text and are processed by Phase 64 if OPT is specified.

ADDRESS INCREMENT

0	1-3	4
80	Value that is to modify an address	Code (3)

ADDRESS REFERENCE

0	1-3	4-6
78	Addressing parameters (4)	Dictionary pointer (5)

BLOCK NUMBER

0	1
C4	Block number

PROCEDURE BASE REGISTER ELEMENT FOR PNs

0	1-2
C8	PN number

PROCEDURE BASE REGISTER ELEMENT FOR GNs

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 type 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.

(3) Code	Meaning
00	LA instruction not to be generated
0E	LA instruction to be generated, using register 14 as base register
0F	LA instruction to be generated, using register 15 as base register.

(4) 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 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 BLLCTR in COMMON  
 2 Subscript cell number  
 3 SBL number assigned from SBLCR in COMMON

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

- ⑤ Bits    Contents  
 0-1        Unused  
 2-14      Dictionary section number  
 15-23     Displacement in section

- 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:

<u>Bits</u>	<u>Contents</u>
0-3	Base register
4-15	Displacement from start of area controlled by base locator

END OF LISTING A-TEXT

0
01

E-TEXT

MESSAGE DEFINITIONS

0	1	2	3-4
00	06	00	Identifying message number

5-6	7
Sequential source card number	(1)

MESSAGE PARAMETERS

0	1	2
00	n	IC-text identifier for parameter (2)

3 through n + 1
Either the actual value to be inserted, or a pointer to the PARTBL field where the value is.

DELIMITER

0
01 (3)

(1) Bits	Contents
0-3	Severity code, as follows: 0000 = W-level (Warning) 0001 = C-level (Conditional) 0010 through = E-level (Error) 0111 1000 = D-level (Disaster)

4-7 Phase number

(2) 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
75	Figurative constant
79	Standard data-name
87	Procedure-name definition
F9	Global Table reference, Type 1
AA	Global Table reference, Type 2
FE	Dictionary pointer (not an IC-text element)

(3) This element is written on SYS004 only if the SYMDMP, STATE, or FLOW option is in effect and the program is segmented. It identifies the end of DATA A-text, DEF-text, and E-text to Phase 60 and the beginning of text that is to be passed to phase 65.



XREF-TEXT

DEF-TEXT ELEMENT FOR DATA-NAME DEFINITION  
OR FILE-NAME DEFINITION

0	1-2	3-5	6	7 to c+6
48	Card number	Pointer to dictionary entry for data-name	c	External-name in EBCDIC

DEF-TEXT ELEMENT FOR VERB DEFINITION

0	1-2	3	4	5	6	7	8 to c+6
48	Number of occurrences	E0	Alpha-betic verb sequence number	00	c	FB	Verb text

DEF-TEXT ELEMENT FOR PROCEDURE-NAME DEFINITION

0	1-2	3-4	5	6 to c+5
4C	PN number	Card number	c	External-name in EBCDIC

REF-TEXT

0-2	3-4
Pointer to dictionary entry for data-name or PN number	Card number (1)

REF-TEXT FOR VERBS

0	1-2	3-4
E0	Verb code (3)	Card number (4)

DELIMITER

0
77
(2)

(1)	<u>Bit</u>	<u>Meaning</u>
	0	0 = Reference is for data-name
	1	1 = Reference is for procedure-name

- (2) This element identifies end of REF-TEXT.
- (3) The alphabetic verb sequence number followed by X'00'.
- (4) The high-order bit will always be off.

DEBUG-TEXT

CARDLOC ELEMENT

0	1-3	4-6
10	COBOL card number ①	Contents of LOCCTR in COMMON when this card was encountered

ENDSEG ELEMENT

0	1-3	4-6
20	Zeros	Contents of LOCCTR in COMMON after processing last verb in a segment or last verb in the program if not segmented

SEGMENT ELEMENT

0	4
30	Priority

DISCONTINUITY ELEMENT

0
40

① The high order bit in the second byte of the card number field is set to 1 if the card precedes a COBOL verb.

DICTIONARY ENTRY FORMATS

This chapter contains diagrams of the formats of the entries in the dictionary. The dictionary is built in storage by phases 11, 22, and 21, respectively. If it exceeds its allotted storage space, the compiler uses SYS001 as a spill file. The dictionary is used by phase 30 to replace names with their dictionary attributes in Procedure IC-text and is then released. It is also used by phase 25 to build the DATATAB and OBODOTAB tables for the Debug File when SYMDMP has been specified. Dictionary handling is performed by the ACCESS routines described in "Appendix A. Table and Dictionary Handling."

The following notes apply to the format diagrams in this chapter:

- The top row of figures shows the number of bytes in the field.
- Shaded areas indicate fields that are present only if the condition they satisfy is present.
- c = Number of bytes in the following field.
- n = Total number of bytes to follow in the entry.
- Individual notes, applying to particular fields, are numbered consecutively with the numbers encircled.

PROCEDURE-NAME (PARAGRAPH) ENTRY

HASH Table Pointer

3
Pointer to dictionary entry for last name with same HASH table value (1)

Basic Fields

1	Variable
c	Name

Attributes

1	2	2	1	1
Count and major code (2)	Characteristics (3)	PN number of this paragraph	Unused	Priority number

PROCEDURE-NAME (SECTION) ENTRY

HASH Table Pointer

3
Pointer to dictionary entry for last name with same HASH table value (1)

Delimiter Pointer

3
Pointer to dictionary entry for next section-name (1)

Basic Fields

1	Variable
c	Name

Attributes

1	2	2	1
Count and major code (2)	Characteristics (3)	PN number of this section	Unused

1	2
Priority number	PN number of next section

FD ENTRY

HASH 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	DTF number	Access method and I/O specifications (5)

1	2	1	1
Position number	Maximum record length	Count and I/O options (6)	Flag byte (7)

2	3
Maximum user label length	(8)

1	1	1	2	1
Secondary DTF number (9)	Secondary DTF number (9)	Secondary DTF number (9)	Block size (9)	Device code (9a)

FD ENTRY FOR VSAM FILES

HASH Table Pointer	Delimiter Pointer	Basic Fields	Attributes				
3	3	1 Variable	1	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	FIB number	Access method and I/O specifications (5a)
			1	2	1		
			Count of entries in IND1TBL for file	Displ. of first entry in IND2TBL	Number of BLs needed for this file		

SD ENTRY

HASH 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 (10)	First BL number	Maximum record length	Minimum record length	Second flag byte (11)

RD ENTRY

HASH 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

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) (12)	Minor code and flags (13)	Addressing parameters for item (14)	Flag byte (15)	Level number and variable information (12) (16)

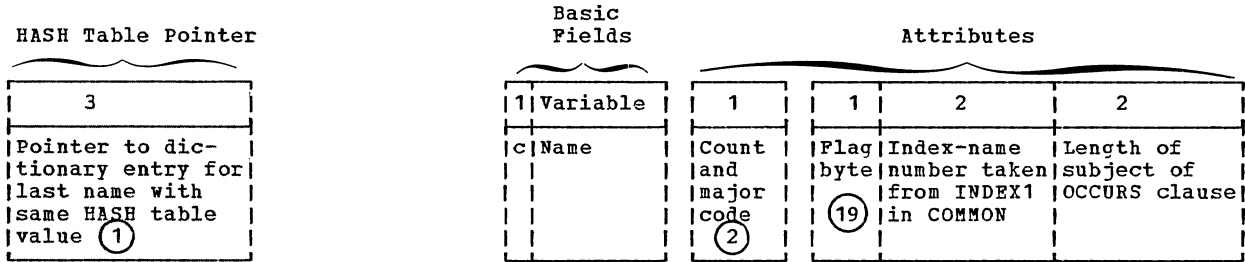
CONDITION-NAME ENTRY

HASH Table Pointer	Basic Fields	Attributes			
3	1 Variable	1	1	3	1
Pointer to dictionary entry for last name with same HASH table value (1)	c Name	Count and major code (2)	0	Pointer to dictionary entry for data item to be tested for condition (1)	Switch (17)

1 Variable
c P1-text element (18)

INDEX-NAME 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 pointer
  - 2-14    Dictionary section number
  - 15-23    Displacement in section

- ② Bits    Contents
- 0-3    Number of bytes in attributes field. If 0, see "Elementary Item with Report Picture" portion of note 16. Count=9 if VSAM file
  - 4-7    1101 = Procedure-name entry
  - 1000 = FD entry
  - 1001 = SD entry
  - 1110 = RD entry
  - 0000 = LD entry under FD
  - 0001 = LD entry under SD
  - 0011 = FD entry (VSAM)
  - 0100 = LD entry in Working-Storage
  - 0101 = LD entry in Linkage Section
  - 0110 = LD entry in Report Section
  - 1100 = Condition-name entry
  - 1111 = Index-name entry

- ③ Bit    Meaning, if on
- 0    Procedure-name
  - 1    Section-name
  - 2    Either name follows TERU in PERFORM...THRU or it follows PERFORM in a PERFORM without THRU.
  - 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 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 referring to such section. Bits 12-15 describe type of section.
  - 12    Declarative error routine
  - 13    Declarative label routine
  - 14    Unused
  - 15    Declarative report section

- ④ Bits    Contents
- 0    1 = Q-routine indication
  - 1-2    00 = Format V
  - 01 = Format F
  - 10 = Format U
  - 11 = Format S
  - 3    Multiple Reel
  - 4-7    Number of BLs needed for this file

⑤	<u>Bits</u>	<u>Meaning</u>
	0-3	Access Method 0001 = DTFCD 0010 = DTFPR 0011 = DTFMT 0100 = DTFSD 0101 = DTFDA 0110 = DTFIS 0111 = DTFDU
	4	1 = OMR (optical mark read) if DTFCD 1 = LOAD file if DTFIS 1 = APPLY WRITE ONLY if DTFMT or DTFSD 1 = IBM extension, if DTFDA, Random Access 1 = ACTUAL KEY, if DTFDA, Sequential Access Otherwise, Unused
	5	1 = Direct file with relative addressing 1 = ASCII, if DTFMT 1 = Double buffered, if DTFIS
	6	1 = 3525 associated file (print)
	7	1 = 3525

⑤a	<u>Bits</u>	<u>Meaning</u>
	0-2	Access mode 100 = ACCESS IS SEQUENTIAL 010 = ACCESS IS RANDOM 001 = ACCESS IS DYNAMIC
	3	Unused
	4-5	Reserved
	6	1 = VSAM INDEXED
	7	1 = VSAM ADDRESSED SEQUENTIAL

⑥	<u>Bits</u>	<u>Contents</u>
	0-3	Number of bytes that follow
	4	1 = SAME RECORD AREA
	5	1 = RANDOM ACCESS 0 = SEQUENTIAL ACCESS
	6-7	00 = standard label 01 = user standard label 10 = nonstandard label 11 = labels omitted

⑦	<u>Bits</u>	<u>Contents</u>
	0	1 = USING or GIVING specified
	1	Unused
	2-4	Carriage control type 000 = none 001 = WRITE AFTER POSITIONING 010 = WRITE AFTER 100 = WRITE BEFORE 110 = mixed
	5	1 = file single-buffered, unblocked or ISAM
	6	1 = CLOSE WITH LOCK
	7	1 = SELECT OPTIONAL

⑧ Dictionary pointer to ACTUAL KEY, if specified. Phase 30 appends idk addressing parameters for the ACTUAL KEY.

⑨ If access method is DTFDA, DTFCD, DTFDR, or DTFIS (ACCESS RANDOM), fields contain zero. Fields also contain zeros, unless OPEN option for a particular access method was specified. Otherwise, fields are as follows:

<u>Access method</u>	<u>Field</u>	<u>OPEN Option</u>
DTFMT	1st	INPUT
	2nd	OUTPUT
	3rd	INPUT REVERSED
DTFSD	1st	INPUT
	2nd	OUTPUT
	3rd	I/O
DTFIS (ACCESS SEQUENTIAL)	1st	RETRIEVE
	2nd	LOAD
	3rd	Contains zeros

⑨a	<u>Code</u>	<u>Device</u>
	80	2560
	40	5424

⑩	<u>Bits</u>	<u>Contents</u>
	0	0
	1-2	00 = Format V or S 01 = Format F 10 = Format U
	3	1 = ASCII collating sequence
	4-7	Number of BLs needed for this file

⑪	<u>Bits</u>	<u>Contents</u>
	0-1	01 = LABEL RECORD STANDARD 10 = LABEL RECORD OMITTED
	2-3	0
	4-7	Number of work units

⑫ During phase 30 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 6 is on, the level is 77. Otherwise, these bits are off.



- |   |  |  |   |
|---|--|--|---|
| <p>⑬ <u>Bits</u>    <u>Code</u>    <u>Operand's Characteristics</u></p> <p>0-3    0000    Error detected for this operand</p> <p>0001    Fixed-length group</p> <p>0010    Alphabetic</p> <p>0011    Alphanumeric</p> <p>0100    Variable-length group</p> <p>0101    Numeric-edited item</p> <p>0110    Sterling report item</p> <p>0111    Usage is index</p> <p>1000    External decimal</p> <p>1001    External floating-point</p> <p>1010    Internal floating-point</p> <p>1011    Binary</p> <p>1100    Internal decimal</p> <p>1101    Sterling nonreport</p> <p>1110    Alphanumeric edited</p> <p>1111    Unused</p> <p>4-5    <u>Code</u>    <u>Subscripts Required</u></p> <p>        00    None</p> <p>        01    1</p> <p>        10    2</p> <p>        11    3</p> <p>6      1 = OCCURS clause in this item</p> <p>7      1 = REDEFINES clause for this item</p> | <p><u>Bits</u>    <u>Field</u>    <u>Meaning</u></p> <p>0-3      i      Type of BL containing base address of area:<br/>                    0000 = BL<br/>                    0001 = BLL<br/>                    0100 = SBL</p> <p>4-15     d      Displacement from base address</p> <p>16-23    k      BL number</p> | <p>⑮ <u>Bits</u>    <u>Meaning, if on</u></p> <p>0        INDEXED BY</p> <p>1        SYNCHRONIZED</p> <p>2        Subject of key</p> <p>3        (If bit 3 = 1)<br/>          0 = DESCENDING<br/>          1 = ASCENDING</p> <p>4        Report WITH CODE</p> <p>6-7     00 = TRAILING<br/>          01 = LEADING<br/>          10 = SEPARATE TRAILING<br/>          11 = SEPARATE LEADING</p> | <p>⑯ <u>Bits</u>    <u>Contents</u></p> <p>0-5     Level number</p> <p>6        1 = 0-routine required</p> <p>7-End   Variable, according to the following chart:</p> |
|---|--|--|---|
- ⑭ Location of item in data area of object module:

16 (Continued)

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 or Usage is index	1 2 3-9 10-17 18-32* 33-41*	1, if PICTURE contains S. Flag bit If bit 2 is on, bits 3 through 9 contain the number of Ps to left of decimal point. If bit 2 is off, they contain the total number of Ps plus 9s to right of decimal point. 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 = Sign plus 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.

\*These bits are present only if the item is in a group with an OCCURS clause. They are not present after phase 30, but at the end of the attributes from the dictionary, 3 bytes are appended indicating either the length of the item which this is subordinate to or the VLC of the item.

16 (Continued)

Characteristic	Bits	Contents
Sterling Non-report Elementary Item	1	0 = BSI shillings 1 = IBM shillings
	2	0 = 1-character pence 1 = 2-character pence
	3	0 = BSI pence 1 = IBM pence
	4- 9	Number of decimal positions to right of V in pence field
	10-14	Number of pound field digits
	15-17	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
	18-25	110 = Unused
111 = Unused		
18-25 or 18-32*	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.	
33-41*	Displacement in section	
42-49	Number of character positions	
Variable-Length Group Item	1- 2	Unused
	3- 5	Number of 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 = 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 58-END	Size of item 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.
<p>*These bits are present only if the item is in a group with an OCCURS clause. They are not present after phase 30, but at the end of the attributes from the dictionary, 3 bytes are appended indicating either the length of the item which this is subordinate to or the VLC of the item.</p> <p>**These bits are not present after phase 30.</p>		

16 (Continued)

Characteristic	Bits	Contents
Numeric Edited Item	1	1 = 2 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 = BLANK WHEN ZERO
	43	1 = * represents all numeric characters
	44	Unused
	45-49	Number of digit places in item
	50-57	Scaling factor
	58-65	Size of item
	66-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. Exception: for CR and DB, first character appears in byte 1, the second in byte 2.
	Elementary Item with Sterling Report PICUTRE	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 = BLANK WHEN ZERO
43		0 = shilling delimiter is D 1 = shilling delimiter is S
44		Same as bit 19 for pounds
45		1 = 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	
<p>*These bits are present only if the item is in a group with an OCCURS clause. They are not present after phase 30, but at the end of the attributes from the dictionary, 3 bytes are appended indicating either the length of the item which this is subordinate to or the VLC of the item.</p> <p>**These bits are not present after phase 30.</p>		

17 If the switch = 1, the next (c) field is followed by a 2-byte field containing the displacement in the VALTRU table of the object of the VALUE clause. If the switch = 0, the rest of the entry is as shown.

Byte 1	Type of Element
32	NUMERIC LITERAL
33	FLOATING-POINT LITERAL
34	ALPHANUMERIC LITERAL
39	ALL Constant

18 The first byte indicates the type of element according to the table below. It is followed by the rest of the P1-text element.

Bit	Meaning, if on
0	Subject is variable length; last field contains VLC number.
1-7	Unused

DEBUG FILE TABLES

When SYMDMP or STATE are specified on the CBL card, phases 25 and 65 build additional tables for debugging purposes. When SYMDMP is specified, all seven tables which are diagrammed in this appendix are built and written on file SYS005. This file may be either on disk or on tape. When STATE is specified, only the PROCTAB and SEGINDX tables are built. These are written by phase 65 in the object module.

The tables are accessed during execution of the program or at abnormal termination of the program by the subroutines of the Symbolic Dump program. For details see the publication IBM DOS/VS COBOL Subroutine Library, Program Logic, Order No. LY28-6424.

The seven 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. 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.

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, and PROCINDX 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 file is single buffered and the address of the buffer is placed by phase 01 in location FIL5BUF in COMMON. Each phase that uses the debug file 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 file.

Figure 63 shows the positions of the tables in the debug file (SYS005). The PROGSUM table contains information about and pointers to the other tables in the file.

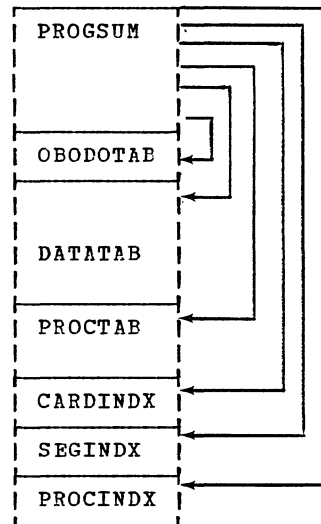


Figure 63. SYS005 (Debug File)

PROGSUM TABLE

The PROGSUM table is the first table on the debug file. It consists of a single fixed-length 108-byte entry and contains information about the program and the debug file itself.

<u>Displ</u>		<u>Field Name</u>	<u>Bytes</u>	<u>Field Description</u>
<u>Dec</u>	<u>Hex</u>			
0	0	PGPROGID	8	PROGRAM-ID
8	8	PGDECLEN	4	Length of Declaratives Section
12	C	PGBL1	4	BL1 address relative to the start of the TGT
16	10	PGBLL1	4	BLL1 address relative to the start of the TGT
20	14	PGSBL1	4	SBL1 address relative to the start of the TGT
24	18	PGDTF1	4	DTF1 address relative to the start of the TGT
28	1C	PGVLC1	4	VLC1 address relative to the start of the TGT
32	20	PGINDEX1	4	INDEX1 address relative to the start of the TGT
36	24	PGENDDTF	4	End of the DTFs relative to the start of the TGT
40	28	PGENDNDX	4	End of the indexes relative to the start of the TGT
44	2C	PGDTDVAD	4	Device address of first block in DATATAB
48	30	PGDTNUM	2	Number of blocks in DATATAB
50	32	PGDTDSP	2	Displacement in the block of the first DATATAB entry
52	34	PGPTDVAD	4	Device address of PROCTAB
56	38	PGCXDVAD	4	Device address of CARDINDEX
60	3C	PGSXDVAD	4	Device address of SEGINDEX
64	40	PGPXDVAD	4	Device address of PROCINDEX
68	44	PGCXNUM	2	Number of entries in CARDINDEX
70	46	PGSXNUM	2	Number of entries in SEGINDEX
72	48	PGPXNUM	2	Number of entries in PROCINDEX
74	4A	PGSXDSP	2	Displacement in the block of the first SEGINDEX entry
76	4C	PGPXDSP	2	Displacement in the block of the first PROCINDEX entry
78	4E	PGFPDSP	2	Displacement of floating-point virtual from the start of the PGT
80	50	PGODONUM	2	Number of bytes in OBODOTAB including the unused bytes at the end of the blocks
82	52	PGHASH	2	Hashed compilation indicator which is matched by the COBOL library subroutine with the one in the DEBUG TABLE in the TGT. The date is hashed from the Communication Region within the Supervisor.
84	54	PGFIB	4	Address of first File Information Block (FIB) relative to start of TGT.
88	58	PGLNGTH	1	Length of PROGSUM table
89	59	PGSLACK	19	Reserved

**Note:** The only fields that may be zero in this table are PGDECLEN, PGODONUM, and PGFPDSP when there is no Declarative Section, or no OCCURS...DEPENDING ON clauses, or no internal floating-point items in the program. 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 the PROGSUM table on the next fullword boundary 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 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.

DATATAB TABLE

The DATATAB table is the third table in the debug file. 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 has the same format for all entries. It varies in length between 7 and 36 bytes. The Variable Attributes field differs for each type of entry and is described on the following pages.

<u>Displ</u>		<u>COUNT-NAME-TYPE FIELD</u>		
<u>Dec</u>	<u>Hex</u>	<u>Field Name</u>	<u>Bytes</u>	<u>Field Description</u>
0	0		1	Count field: Number of bytes (c) in name field
1	1		c	Name field: Number of varies between 1 and 30
1+c	1+c		1	Count field: Number of bytes in remainder of entry
2+c	2+c	CARDNUM	3	Card number where name is defined (contains zeros for RENAMES items)
5+c	5+c	MAJMIN	1	Type of entry

<u>Bit</u>		<u>Meaning</u>
<u>Bits</u>	<u>Settings</u>	
0-3	1000XXXX	FD entry (non-VSAM)
	1011XXXX	FD entry (VSAM)
	1001XXXX	SD entry
	1110XXXX	RD entry
	1111XXXX	Index-name
	0000XXXX	Level description under FD
	0001XXXX	Level description under SD
	0110XXXX	Level description under RD
	0100XXXX	Level description in Working-Storage
	0101XXXX	Level description in Linkage
4-7	XXXX0001	Fixed length group
	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
	XXXX1011	Binary
	XXXX1100	Internal decimal
	XXXX1101	Sterling non-report
	XXXX1110	Alphanumeric edited
XXXX1111	RENAMES (level 66)	



VARIABLE ATTRIBUTES FIELD

<u>Displ</u>	<u>Dec</u>	<u>Hex</u>	<u>Field Name</u>	<u>Eytes</u>	<u>Field Description</u>
--------------	------------	------------	-------------------	--------------	--------------------------

SD item: There are no variable attributes for an SD entry.

RFNAMES item (level 66):

<u>Dec</u>	<u>Hex</u>	<u>Field Name</u>	<u>Eytes</u>	<u>Field Description</u>
6+c	6+c	RENAMES	1	

<u>Bit</u>	<u>Setting</u>	<u>Meaning</u>
7	XXXXXXX1	Next DATATAP entry RENAMES the same item as this one does
	XXXXXXX0	This is the last (or only) item renaming an item.

INDEX name:

<u>Dec</u>	<u>Hex</u>	<u>Field Name</u>	<u>Eytes</u>	<u>Field Description</u>
6+c	6+c	INDXCELL	2	Index cell number in TGT

FD item (other than VSAM):

<u>Dec</u>	<u>Hex</u>	<u>Field Name</u>	<u>Eytes</u>	<u>Field Description</u>
6+c	6+c	DTFNUM	1	DTF number
7+c	7+c	ACCESSPLG	1	Access method

<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>
0-3	0001XXXX	DTFCD
	0010XXXX	DTFPR
	0011XXXX	DTFMT
	0100XXXX	DTFSD
	0101XXXX	DTFDA
	0110XXXX	DTFIS
7	XXXXXXX1	Sequential access method
	XXXXXXX0	Random access method

FD item (VSAM):

<u>Dec</u>	<u>Hex</u>	<u>Field Name</u>	<u>Eytes</u>	<u>Field Description</u>
6+c	6+c	FIB	1	FIB number
7+c	7+c	ORGACC	1	

<u>Bit</u>	<u>Meaning</u>
0	Access is sequential
1	Access is random
2	Access is dynamic
3	Unused
4	Reserved
5	Reserved
6	Organization is indexed
7	Organization is sequential

RD item:

<u>Dec</u>	<u>Hex</u>	<u>Field Name</u>	<u>Eytes</u>	<u>Field Description</u>
6+c	6+c	LINECTR	3	Addressing parameters of line counter

<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>Dec</u>	<u>Hex</u>	<u>Field Name</u>	<u>Eytes</u>	<u>Field Description</u>
9+c	9+c	PAGECTR	3	Addressing parameters of page counter (same form as addressing parameters above)

Level Description item:

Variable attributes for level description items are divided into two portions: (1) the type-dependent portion, (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:

6+c	6+c	IDKFLD	3	Addressing parameters (same form as above)
9+c	9+c	LVLRDEFN	3	

	Bit		
	<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:

6+c	6+c		3	Addressing parameters (same form as above)
9+c	9+c	MAXSIZE	3	

	Bit		
	<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)

12+c	C+c	VLCNUM	2
------	-----	--------	---

	Bit		
	<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>
	0	1XXXXXXXX	ODO Master
	1-3		(Unused)
	4-15		VLC number

ELEMENTARY, ALPHARETIC, ALPHANUMERIC, REPORT, EDITED, STERLING, EXTERNAL FLOATING POINT:

6+c	6+c		3	Addressing parameters (same form as above)
9+c	9+c	JUSTRGT	3	

	Bit		
	<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:

6+c	6+c		3	Addressing parameters (same form as above)
9+c	9+c	FLPTYPE	1	

	Bit		
	<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>
	0-5		Normalized level number
	6	XXXXXX1X	REDEFINFS
	7	XXXXXXX0	COMP-1
		XXXXXXX1	COMP-2

10+c	A+c		2	(Unused)
------	-----	--	---	----------

BINARY, INDEX, INTERNAL DECIMAL EXTERNAL DECIMAL:

6+c	6+c		3	Addressing parameters (same form as above)
9+c	9+c	NUMINFO1	1	

<u>Bit</u>	<u>Settings</u>	<u>Meaning</u>
0-5		Normalized level number
6	XXXXXX1X	REDEFINES
7	XXXXXXX1	S in PICTURE
0	1XXXXXXXX	Leading sign
	0XXXXXXXX	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 bit 3 equals 1, number of Ps in picture
14-15		(Unused)

(2) Subscripting Information Portion:

This portion of the Variable Attributes section begins immediately after the type-dependent portion.

It ranges in size from 1 byte 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

Bit	Settings	Meaning
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 or the middle inclusive group of 3 or elementary group contains an ODO.
6	XXXXXX1X	Item requires 3 subscripts.
7	XXXXXXX1	OCCURS clause connected with the least inclusive group of three or elementary item contains an ODO.

1 VLC information

Bit	Settings	Meaning
0	1XXXXXXX	Most inclusive or only group of 3
1	X1XXXXXX	Less inclusive group of 2 or middle inclusive group of 3
2	XX1XXXXX	Least inclusive group of 3

If any of these bits equals 1, bytes 2 and 3 of the group length information for the associated group contain a VLC number rather than the length of the group.

1st subscript (if present)	[ 2	Number of occurrences (Maximum number if ODO) specified in OCCURS clause governing this item.
	[ 2	Displacement of next occurrence governed by OCCURS clause (See note)
(most inclusive with OCCURS)	[ 2	Number of occurrences (as above)
	[ 2	Displacement of next occurrence governed by OCCURS
2nd subscript (if present)	[ 2	Number of occurrences (as above)
	[ 2	Displacement of next occurrence governed by OCCURS
3rd subscript (if present)	[ 2	Number of occurrences (as above)
	[ 2	Displacement of next occurrence governed by OCCURS
(least inclusive with OCCURS)	[ 2	Displacement of next occurrence governed by OCCURS

{ 1st subscript with ODO (if present)	[ 2	OBODOTAB pointer for most inclusive group or elementary item containing an ODO	]			
		<table border="0"> <thead> <tr> <th><u>Bit</u></th> <th><u>Contents</u></th> </tr> </thead> <tbody> <tr> <td>0-8</td> <td>Relative block number in OBODOTAB</td> </tr> <tr> <td>9-15</td> <td>Displacement within block (in fullwords)</td> </tr> </tbody> </table>		<u>Bit</u>	<u>Contents</u>	0-8
<u>Bit</u>	<u>Contents</u>					
0-8	Relative block number in OBODOTAB					
9-15	Displacement within 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)	]			

Note: All subscript length information precedes any OBODOTAB pointers.

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 has a card and/or 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>Displ</u>		<u>Field Name</u>	<u>Bytes</u>	<u>Field Description</u>						
<u>Dec</u>	<u>Hex</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>Displ</u>		<u>Field Name</u>	<u>Bytes</u>	<u>Field Description</u>						
<u>Dec</u>	<u>Hex</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 two levels:

1. Ascending priority number
2. Ascending fragment number within a priority

<u>Displ</u>		<u>Field Name</u>	<u>Bytes</u>	<u>Field Description</u>
<u>Dec</u>	<u>Hex</u>			
0	0	SXPRIOR	1	Priority number
1	1	SXRELAD	3	Address of this fragment relative to the beginning of the segment
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 CARDINDX or SEGINDX table.

<u>Displ</u>		<u>Field Name</u>	<u>Bytes</u>	<u>Field Description</u>
<u>Dec</u>	<u>Hex</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	Relative address of instructions for this entry within segment to which it belongs.
6	6	PXDEVADR	4	Device address of PROCTAB table block related to this entry.

VSAM FILE INFORMATION BLOCK (FIB)

The file information block, a portion of the completed object module, is used at execution time by the ILBDVOC0 and ILBDVIO COBOL library subroutines for processing input/output verbs used with VSAM files. The FIB is built by phase 21 and completed by the ILBDVOC0 subroutine.

Fixed Portion:

<u>Displacement</u>		<u>Field</u>	<u>No. of Bytes</u>	<u>Description</u>
<u>Hex</u>	<u>Decimal</u>			
0	0	INAMED	7	External name
7	7	INAMECB	1	External name
8	8	IDEVICE	1	Device class and number
9	9	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=S
	IORGSO	1000 0100	SEQUENTIAL	Code=S
	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
	IORGRLO	0010 0100	RELATIVE	Code = R
	IORGDIR	0010 0010	DIRECT	Code = D
	IORGDIW	0010 1000	DIRECT (WRITE/REWRITE)	Code = W
	IORGVSAM	0000 1000	VSAM File	

A 10 IACCESS 1 ACCESS MODE

Code:

<u>Bits</u>	<u>Equate Name</u>	<u>Bit Settings</u>	<u>Meaning</u>
0-7	IACCSEQ	1000 0000	SEQUENTIAL
	IACCRAN	0100 0000	RANDOM
	IACCDYN	0010 0000	DYNAMIC

B 11 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

C 12 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 AREA specified
	ISLBOMIT	0000 1000	LABEL RECORDS ARE OMITTED
	ISLBSTAN	0000 0100	LABEL RECORDS ARE STANDARD
	ISLBUSER	0000 0010	LABEL RECORDS ARE dataname

0D 13 ISW2 1 Miscellaneous switches

Code:

0-7	ISADVAN	1000 0000	WRITE ADVANCING
	ISPOSIT	0100 0000	WRITE POSITIONING
	ISAFTEP	0010 0000	WRITE AFTER
	ISBEFORE	0001 0000	WRITE BEFORE
	ISNOSPAC	0000 1000	WRITE WITHOUT SPACING



Displacement			No. of	
<u>Hex</u>	<u>Decimal</u>	<u>Field</u>	<u>Bytes</u>	<u>Description</u>
0E	14	ISW3	1	Unused
0F	15	ISW4	1	Unused
10	16	IAPPLY1	1	APPLY statements
11	17	IAPPLY2	1	APPLY statements
12	18	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 set in 'BLOCK CONTAINS 1 RECORD'.
14	20	IRECLEN	2	Number of bytes in longest 01-entry
16	22	IRECDBL	2	Displacement in TGT of record's first base locator cell
18	24	IRECNBL	1	Number of base locators for RECORD AREA
19	25	IRESERVE	1	Reserve integer areas
1A	26	ISTATDBL	2	Displacement in TGT of base locator for STATUS data-name
1C	28	ISTATDDN	2	Displacement from base locator of STATUS data-name
1E	30	ISTATLDN	2	Length of STATUS data-name
20	32	IKEYISW	1	Miscellaneous switches
21	33	IKEYNO	1	Number of entries in key list
22	34	IKFYFNTL	2	Length of each entry in key list
24	36	IPSWISW	1	Miscellaneous switches
25	37	IPSWNO	1	Number of entries in password list
26	38	IPSWENTL	2	Length of each entry in password list
28	40		14	Reserved
36	54		2	Slack bytes
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 for first Key list list entry.
48	72		4	Reserved

Variable Length Portion:

Supplementary Information for Miscellaneous Clauses (one for each clause):

Displacement		Field	No. of Bytes	Description
Hex	Decimal			
0	0	IMSW1	2	Switch bytes
Code:				
			<u>Bits</u>	<u>Equate Name</u> <u>Bit Settings</u> <u>Meaning</u>
			0-7	IMRREOV    1000 0000    RERUN at end of volume
			8-15	Unused
2	2	IRERUNI	4	RERUNN integer (Field contains zeros if RERUN not specified)
6	6	IRERUNN	8	External-name of RERUN clause
E	14			Slack bytes

Indexed Record Key List:

0	0	IRKEYLDN	2	Length of RECORD KEY data-name
2	2	IRKEYDDN	2	Displacement in record of RECORD KEY
				Record Key information follows for each ALTERNATE RECORD KEY specified

SECTION 6. DIAGNOSTIC AIDS

This chapter provides information about compiler error messages and provides a few diagnostic aids for use in case a compiler error, rather than a user error, appears. A compiler error may produce one of two results: an abnormal termination while the compiler is still in storage, or erroneous output from a completed compilation.

Note: The compiling program-name, its version numbers, its modification number, and the Program-ID can be found at the end of the INIT1 routine (see Appendix B). INIT1 is at the end of the object module listing.

ERROR MESSAGE LISTING

A complete listing of all the error messages produced by the compiler can be obtained by compiling a program with a PROGRAM-ID of ERRMSG. The listing consists of:

- Compiler identification and level
- Brief description of the remainder of the listing
- Description of listing conventions
- Action codes and meanings
- General procedure in recurring problem situations
- Error messages in the following format:

```
-- Message number
-- Severity level
-- Action code
-- Message text
```

The messages are listed by phases of the compiler in the following order: 10, 11, 20, 21, 22, 30, 40, 50, 60, 62, 63, 64, 00. The completed listing can be inserted in an 8 1/2" by 11" binder for convenient reference.

DUMP PRODUCED FOR DISASTER LEVEL ERROR MESSAGES

Under severe circumstances the compiler determines that processing during a phase cannot continue. The phase issues a Disaster-level (D-level) message on SYSLSLST and initiates a storage dump. Compilation is abandoned and control is returned to the operating system.

The dump indicates which phase of the compiler was executing at the time of the D-level circumstance. The contents of registers 0 through 15 at the time of the last entry to the TAMEL routines in phase 00 may be found in the 16 fullwords starting at label TBSAV1 in the load module ILACBL00.

ABNORMAL TERMINATION DURING COMPILATION

If the compiler terminates execution abnormally, the resulting dump can be used to locate some information from the data in storage at the time. If certain input/output error messages were printed, additional information is available.

LOCATION OF INFORMATION IN STORAGE

The programmer can determine from a dump the current processing phase in storage, the current record, the status of tables and buffers already in use, and the registers saved during the previous calling process.

Current Phase

If the compiler terminates execution while a processing phase (05, 06, 07, 08, 10, 11, 12, 20, 22, 21, 25, 30, 40, 50, 51, 60, 61, 62, 63, 64, 65, or 70) is in storage, the name of the phase can be found in the Phase 00 location LINKNAME. LINKNAME contains an 8-byte name in the form FCOBOLxx, where xx represents the phase number.

Each of the five words located 15 statements after LINKNAME in storage contains an address. These addresses point to the locations where one or more phases

are loaded. The five address constants, in order, are:

1. Address of Phase 01.
2. Address of Phases 10, 11, and 12.
3. Address of Phases 20, 21, 22, 25, and 30.
4. Address of Phases 40, 50, 51, 60, 62, 63, 64, 65, and 61.
5. Address of Phase 70.

Phase 00 is loaded immediately after the LOS/VS System Supervisor.

Current Record

Unless either phase 20 or phase 21 is executing, the record that is being processed at the time of the error can be related to a statement in the listing through a location in COMMON named CURCRD, which contains the current generated card number. (CURCRD is located at relative address 26C. See "Section 5. Data Areas" for more information on COMMON.) If either phase 20 or phase 21 is executing, the generated card number can be found in the CARDNO field or the CARDNUM field, respectively. If there is no source 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 below under "Compiler Buffers and Their Contents."

Tables Used by Phases

The status of the tables built or referenced by the current phase may reveal how much processing the phase had done before the dump occurred. Tables currently being used by a given phase can be located by the following steps:

1. The Table Information Block (TIB) number for any table handled by the TAMEK routines can be found in "Section 4. Directory" (listed by phase) or in "Section 5. Data Areas" (listed alphabetically by table name).
2. Add the phase 00 load address to the displacement (shown in the listing of phase 00) for the particular TIB

number. The result is the address of the proper TIB.

3. The second, third, and fourth bytes in the TIB contain the address of the Table Area Management Map (TAMM) for the table.
4. 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 the TAMEK routines is shown in "Section 5. Data Areas." See also "Appendix A. Table and Dictionary Handling."

Compiler Buffers and Their Contents

When a processing phase requests an input/output operation via phase 00, register 9 contains the address of an entry for the file within the phase 00 POINT table. Figure 64 shows the contents of a POINT table entry. Each consists of two bytes, with the low-order bits of each byte containing the buffer numbers. If a file 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 file in the current phase.

Bits	Contents
0-3	X'0' or X'F'
4-7	buffer number
8-11	X'0'
12-15	buffer number

Figure 64. POINT Table Entry Format

The buffer number can then be used to locate the associated Buffer Control Block (BCB). BCBs are contained in an area starting at BUFCNLS in phase 00, as shown in Figure 65. The format of a BCB is shown in Figure 66. For SYSIPT, SYSLST, SYSPCH, and SYSLNK, the buffer addresses are contained in the phase 00 locations IPTBUF, LSTBUF, PCHBUF, and LNKEUF, respectively.

Location	Control Block
BUFCNLS+0	Buffer 1BCB
BUFCNLS+8	Buffer 2BCB
BUFCNLS+16	Buffer 3BCB
BUFCNLS+24	Buffer 4BCB
BUFCNLS+32	Buffer 5BCB
BUFCNLS+40	Buffer 6BCB

Figure 65. Buffer Control Blocks for Buffers 1-6

Bytes	1	3	2	2
Contents	X'00'	Address of Buffer	Bytes used so far for GET or PUT	Length of Buffer

Figure 66. Buffer Control Block Format

**Note:** The above does not apply to file SYS005. Phases 25 and 65 perform buffer management processing for SYS005.

Register Usage

When linking to another phase, phase 00 passes a pointer in register 1 to a parameter list starting at location COSPARM in phase 00. The first word of this list always contains the address of COS, another location in phase 00. This address is used by the other phases in two ways: as the point in phase 00 to which control should be passed for an input/output request and as the base address for the COMMON DSECT used in the other phases. General register usage for each phase is shown in Figure 67.

PHASE 00 (ILACBL00)	
Register	Use
0	Contains COS address at entry to COS; contains buffer address at exit from READ routine
1	Used to return to calling phase; otherwise, work.  Note: Registers 0 & 1 are destroyed by phase 00 during calls to COS.
2	Address of data for a WRITE.
3	Length of data for a WRITE.
4	Work; Address of TRMNATE CSECT
5	Base register for ILACBL00 CSECT
6	Points to buffer control block for logical I/O
7	Points to buffer control block for physical I/O
8	Return address-2 (Linkage parameters)
9	Address of file pointers (Point table) for the file on which I/O is currently being performed.
10	(File number-1) * 4; used to index into BUFCNLS and the table of DTF ADCONS.
11	Linkage; base register for PHOTBDIC CSECT
12	Linkage; base register for TBDATA CSECT
13	Save area pointer (from calling phase)
14	Linkage.
15	Linkage; base for PHOTBST1

Figure 67. Compiler Register Usage

PHASE 01 (ILACBL01)	
Register	Use
0	Linkage to phase 00; work in CSECT ILA002
1	Contains address of phase 00 parameters when phase 01 is called; work in CSECT ILA002
2-3	Work
4	Work; input pointer in CSECT ILA002
5	Base for CSECT ILACBL01; base for CSECT ILA002 (COPY/BASIS)
6	Base for DATA; base for CSECT ILA002 (COPY/BASIS)
7	Work; base for CSECT ILA002 (COPY/BASIS)
8	Highest available storage address; linkage in CSECT ILA002
9	Base for CSECT ILA002 (COPY/BASIS)
10	Address of system's communication region; work in CSECT ILA002
11	Address of COS; work in CSECT ILA002
12	Address of phase 00 parameter list; work in CSECT ILA002
13	Address of phase 01 save area
14	Internal linkage (SCAN routine); work in CSECT ILA002
15	Internal linkage (SCAN routine); address of COS in CSECT ILA002

PHASE 05 (ILACBL05)	
Register	Use
0	Linkage to phase 00; work
1	Address of COS; work
2-9	Work
10-12	Base
13	Work area DSECT base
14-15	Internal linkage

PHASE 06 (ILACBL06)	
Register	Use
0	Linkage to phase 00; work
1	Address of COS; work
2-9	Work
10-12	Base
13	Work area DSECT base
14-15	Internal linkage

PHASE 07 (ILACBL07)	
Register	Use
0	Linkage to phase 00; work
1	Address of COS; work
2-9	Work
10-12	Base
13	Work area DSECT base
14-15	Internal linkage

PHASE 08 (ILACBL08)	
Register	Use
0	Linkage to phase 00; work
1	Address of COS; work
2-9	Work
10-12	Base
13	Work area DSECT base
14-15	Internal linkage

PHASE 20 (ILACBL20)	
Register	Use
0-6	Work
7	Address of input buffer area
8-10	Work
11	Address of COMMON, except during ACCMET routine
12	Work
13	Address of save area
14	Branching
15	Base for all routines

PHASE 11 (ILACBL11)	
Register	Use
0	Work
1	Address of COS in phase 00
2-7	Work
8	Base of IEQ109 (EXHSVB)
9	Base of IEQ107 (LETTER)
10	Base of IEQ104 (COBWRD)
11	Base of IEQ103 (GETWD)
12	Base of IEQ102 (IDDIV1)
13	Base of IEQ101 (PH1SAV), pointing directly to the phase 11 Save Area in that CSECT
14	Linkage
15	Linkage to TAMER Addressability to IEQ108 (PDSCN) and temporary addressability to IEQ116 (VARPQ) and to IEQ106 (UNLVSN)





PHASE 21 (ILACBL21)	
Register	Use
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 COS in phase 00
12	Base of PERMCODE
13	Pointer to SAVEAREA
14, 15	Linkage

PHASE 25 (ILACBL25)	
Register	Use
0-8	Work
9	Base register for SYMDICT DSECT
10	Address of COS in phase 00
11,12	Permanent base of first 2 CSECTS
13	Permanent base of data CSECT
14	Linkage
15	Linkage, base of access routines

PHASE 22 (ILACBL22)	
Register	Use
0-6	Work
7	Address of input buffer area
8-10	Work
11	Address of COMMON, except during ACCMET routine.
12	Work
13	Address of save area
14	Branching
15	Base for all routines

PHASE 30 (ILACBL30)	
Register	Use
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	Permanent base for the CSECT containing the following routines:  GLORET PH5CTRL READ EOF GENOP SEARCH QUALIF CORRTN
8	Permanent base for the CSECT containing the following routines:  GLOSRY ENTRPD TSTWRD ERROR GETNXT GETALL GENDAT COPYIN COPREN ISTDLM ISTBCD GETLVL COUNT DESCUP ISPTR
9	Permanent base for phase 30 data area

PHASE 40 (ILACBL40)	
Register	Use
0-6	Work; used by verb analyzers, subroutines, and phase controller
7-11	Permanent bases for the nonverb analyzer routines:  phase controller subroutines constants data area
12	Points to current contents of input buffer
13	Unused
14	Branching
15	Temporary base register for the verb analyzer currently in control Linkage to TAMER

PHASE 50 (ILACBL50)	
Register	Use
0-6	Work
7	Base for 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

PHASE 51 (ILACBL51)	
Register	Use
0-6	Work
7	Same as 50
8	Work
9	Base register for each verb analyzer
10	Base register for each verb analyzer
11-15	Base register for each verb analyzer

PHASE 60 (ILACBL60)	
Register	Use
0-3	Work
4	Input for all texts
5-7	Work
8	Base for the following CSECTs: 1--performs phase initialization (switches, work areas, etc.); processes the TGT 3--processes Procedure A-text
9	Points to COMMON in phase 00
10	Base for the following CSECTs: 2--processes Optimization A-text 4--processes Procedure A-text and Data A-text; generates initialization routines
11	Work
12	Base for CSECT containing constants (IEQ605)
13	Points to phase 60 save area
14	Linkage to subroutines
15	Linkage to subroutines

PHASE 62 (ILACBL62)	
Register	Use
0-3	Work
4	Input for all texts
5-7	Work
8	Base for the following CSECTs: 1--performs phase initialization (switches, work areas, etc.); processes the TGT 3--processes Procedure A-text
9	Points to COMMON in phase 00
10	Base for the following CSECTs: 2--processes Optimization A-text 4--processes Procedure A-text and Data A-text; generates initialization routines
11	Work
12	Base for CSECT containing constant (IEQ625)
13	Points to phase 62 save area
14	Linkage to subroutines
15	Linkage to subroutines

PHASE 63 (ILACBL63)	
Register	Use
0-10	Work
11-13	Base registers
14	Return for internal subroutines
15	Address of internal subroutines

PHASE 64 (ILACBL64)	
Register	Use
0-3	Work
4	Input for all texts
5-7	Work
8	Base for the following CSECTs: 1--performs phase initialization (switches, work areas, etc.); processes the TGT 3--processes Procedure A-text
9	Points to COMMON in phase 00
10	Base for the following CSECTs: 2--processes Optimization A-text 4--processes Procedure A-text and Data A-text; generates initialization routines
11	Work
12	Base for CSECT containing constants (IEQ645)
13	Points to phase 64 save area
14	Linkage to subroutines
15	Linkage to subroutines

PHASE 65 (ILACBL65)	
Register	Use
0-3	Work
4	Input for Debug-text and for generating TXTCRD DSECT
5-7	Work
8	Base for CSECT ILA651
9	Points to COMMON in phase 00
10	Base for CSECT ILA652
11-12	Work
13	Points to phase 65 save area; base for phase 65 constant CSECT ILA653
14-15	Linkage to subroutines

PHASE 61 (ILACBL61)	
Register	Use
0	Work; return address when phase 61 has branched to phase 00
1	Work; base for input DSECT
2	Work; pointer to data record being formatted
3-9	Work
10	Points to COMMON in phase 00
11	Base for CSECT ILA6102
12	Base for CSECT ILA6101
13	Points to phase 61 save area
14	Return address for internal subroutines
15	Work; address of internal subroutines

PHASE 70 (ILACBL70)	
Register	Use
0-3	Work
4	Input for E-text
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
11-12	Work
13	Points to phase 70 save area
14	Linkage to subroutines
15	Base for PUT, CONVERT, GET, STRING, and XPRIME routines

PHASE 80 (ILACBL80-8D)	
Register	Use
0	Work
1	Base for ILACBL80; work
2	DTF pointer for ILACBL80; work
3	DTF pointer for ILACBL80; work
4	DTF pointer for ILACBL80; base for ILACBL8C; work
5	Base for ILACBL8C; work
6	Internal link for ILACBL8B, ILACBL8C, and ILACBL8D; work
7-9	Work
10	DTF pointer for ILACBL84, ILACBL86, and ILACBL8D
11	DTF pointer for ILACBL8D; work
12	Points to FIPSVT (FIPS vector table)
13	Save
14	Link
15	Base

## Register Saving

When phase 00 gets control from the system, it places the address of the DOS control program's save area in location MYSAVE+4. When it is called by another compiler phase, phase 00 places the address of the calling phase's save area in SAVER13. It puts the address of its own save area (MYSAVE) in register 13.

The calling phase's registers can be located by adding decimal 12 to the address contained in SAVER13. This locates register 14, followed by 15, etc. The following registers have significance in connection with the call:

Register 0: Contains the address of the X and Y parameters of the linkage request. (See "Processing Between Phases" in the chapter on phase 00.) These parameters can be used with Figure 11 in the phase 00 chapter to verify that the calling phase is making a legitimate request.

Note: Register 10 of phase 00 should contain the file number code (Y parameter minus 1) multiplied by 4.

Register 2: Contains the address within the calling phase from which phase 00 is to write if the X parameter above is a request for a PUT. It can be used to determine that the indicated area is within the range of the storage allocated to the calling phase.

Register 3: Contains the length, in bytes, of the data to be written if a PUT has been requested. The number must be less than 256, 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. 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. For the complete calling sequence, including parameters, see "Appendix A. Table and Dictionary Handling." The MASTAM, also described in that chapter, can be found at location TBMASAM and should be investigated to determine whether the table and dictionary areas are being properly used.

## INPUT/OUTPUT ERROR MESSAGES

Phase 01 generates messages if it discovers error conditions during initialization. The system handles directly all input/output errors.

## ERRONEOUS COMPILER OUTPUT

If the compilation terminates normally (with the desired options in effect) and if the object module nevertheless executes incorrectly, the source program should be checked first for mistakes in logic, language, data formats, etc. Using the SYMDMP, STATE, and FLOW compiler options can help in determining mistakes. For information on the use of these options, see the publication IBM DOS/VS COBOL Compiler and Library Programmer's Guide, Order No. SC28-6478. If a compiler error is still suspected, the information below can help to pinpoint the problem.

Note: The LIST and LISTX compiler options, along with the linkage editor's MAP option, are invaluable under these conditions. The SYM, XREF or SXREF and FLAGW compiler options are also useful.

## STORAGE LAYOUT

An example of the general storage usage for a COBOL program being executed in the background area is given in Figure 68. The Memory Map printed as a result of the LISTX option contains the relative addresses of the TGT fields, the literal pool, PGT fields, the register assignments, the instructions generated from the Procedure Division, and the INIT2, INIT3, and INIT1 routines, in that order. (See "Appendix B Object Module" for a discussion of these fields.) The absolute addresses can then be found with the assistance of the phase map (see "Linkage Editor Phase Map" in this chapter).

CONTROL PROGRAM	Permanent storage locations used by CPU; Communication Region; Supervisor Nucleus; I/O Units Control Tables; and Transient Area	
BACK-GROUND	OBJECT MODULE	INIT1 Working-Storage DTFs and BUFFERS TGT PGT Literals Report Writer Procedure Division (Priority less than segment limit) Q-routines INIT2 INIT3 Transient Area (Nonresident Segments)
	*	
	LLOCS Modules COBOL Library Subroutines	
FORE-GROUNDS II & I		
*The object module is not always first in its partition.		

Figure 68. Example of Storage Usage During Execution

beginning of the object module as given in the linkage editor map.

4. Allowing one fullword per DTFADR cell, count off cells from the starting address found in Step 3, using the order determined in Step 1, to locate the desired DTFADR cell.
5. The DTFADR cell contains the absolute address of the desired DTF.

Note: The procedure for locating a secondary DTF is essentially the same, the only differences being that the SUBDTF address cells pointed to by the PGT are used and that the order of the cells is Input, Output, I-O, or Input Reversed.

LOCATING DATA

The location assigned to a given data-name may similarly be found by using the BL number and displacement given for that entry in the GLOSSARY, and locating the appropriate one-word BL cell in the TGT. The hexadecimal sum of the GLOSSARY displacement and the contents of the cell should give the relative address of the desired area. This can then be converted to an absolute address as above.

LOCATING A DTF

A particular DTF may be located in an execution-time dump as follows:

1. Determine the order of the DTF address (DTFADR) cells in the TGT from the DTF numbers shown for each file-name in the GLOSSARY.

Note: Since the order is the same as the order of FDs in the Data Division, it can be determined from the source program whether the SYM option was not used (that is, no glossary was printed).

2. Determine the relative starting address of the block of DTFADR cells from the TGT listing in the Memory Map.
3. Calculate the absolute starting address of the block by adding the hexadecimal relocation factor for the

REGISTER USAGE

The compiler assigns registers for use at execution time according to the general rules indicated in Figure 69. When OPT is specified, the register assignment is that indicated in Figure 70. In addition, the LISTX or CLIST or SYM option causes the permanent register assignments for the data areas to be printed in the Memory Map, 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 69. Register Usage at Execution

Register	Assignment
0-5	Work
6-9	Assigned in the following order: 1 TGT or PGT OVERFLOW CELLS 2 Most used BLs or BLLs
10	PGT OVERFLOW if another OVERFLOW results after allocation of PROCEDURE BLOCK CELLS or next most used BL or BLL
11	Procedure Block cell
12	PGT
13	TGT
14-15	Temporary base registers

Figure 70. Register Usage at Execution when OPT is Specified

ERROR MESSAGES

All error and warning messages listed during compilation should be examined. (The warning messages will be listed only if the FLAGW option is in effect.) If any message seems inappropriate, it can sometimes be traced back to the originating phase by examining the message identification code. The format is as follows:

ILAnxxxI-S

where:

- n is the first digit of the phase number
- xxx represents the assigned identifying numbers for the message
- s indicates the severity code (W, C, E, or D)

Several phases can begin with the same digit (for example, phases 10, 11, and 12) but no further general distinction is made between the message numbers assigned by the various phases.

LINKAGE EDITOR PHASE MAP

If the MAP option is in effect, the linkage editor produces a phase map showing

the absolute addresses of all entry points of all CSECTS in the phase. Figure 71 is an example of a phase map.

In the figure, ILBDSAE0, a COBOL library subroutine, handles sequential access input/output errors and ILBDDSP0 handles DISPLAY verbs. IJJCPD1 is a DOS logic module which is present whenever a DISPLAY or an ACCEPT verb is used. ILBDMNS0 is a 1-byte COBOL library subroutine which is always present and which provides a flag indicating whether a subprogram or a main program is executing.

DIAGNOSTIC ASSISTANCE

When you (the CE) telephone 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

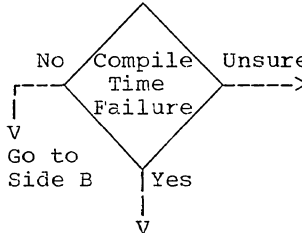
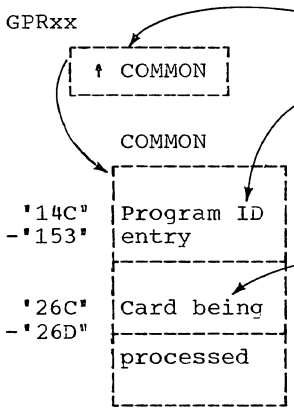


PHASE	XFR-AD	LOCORE	HICORE	DSK-AD	ESD TYPE	LABEL	LOADED	REL-FR
PHASE***	002800	002800	003EB3	5D 05 2	CSECT	EXAMPLE	002800	002800
					CSECT	ILBDSAE0	003E08	003E08
					ENTRY	ILBDSAE1	003E1E	
					CSECT	ILBDMNS0	003E00	003E00
					CSECT	ILBDDSP0	003630	003630
					CSECT	IJJCPD1	003468	003468
					ENTRY	IJJCPD1N	003468	
					* ENTRY	IJJCPD3	003468	

Figure 71. Example of a Phase Map



COBOL Abstract Worksheet (Side A)

Search Item	Search Argument	How to Find Argument
<p>1. Component ID: 5746-CB-xxx where xxx is the program product version level</p>	<p>1. 5746-CB-xxx</p>	<p>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.</p>
<p>2. Keyword entry (use as indicated):</p>	 <p>2. CMPL</p>	<p>If unsure, check the JCL to see which step was being executed at the time of the failure.</p>
<p>3. Type of failure: one of the following:</p> <p>abend = ABENDxxx wait = WAIT loop = LOOP msg. = MSGILAXxxxx</p>	<p>3.</p>	
<p>4. Failing phase: ILACBLxx, where xx is the compiler phase number.</p>	<p>4. ILACBLxx</p>	<p>Location LINKNAME in phase 00 contains a name in the form FCOBOLxx, where xx is the compiler phase number.</p>
<p>5. COBOL verb being processed: (e.g., ADD, MOVE, GO)</p>		<p>A. Find the register used by the failing phase (argument 4 above) to point to COMMON; see "Register Usage" in "Section 6. Diagnostic Aids."<sup>1</sup></p> <p>B. Obtain the address of COMMON from the register.</p> <p>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).</p> <p>D. Find card number at COMMON address + X'26C" (e.g., 800030 would indicate card 48 because X'30" = decimal 48).</p> <p>E. Find the indicated card in the source program and determine the COBOL verb being processed.</p>
<p><sup>1</sup>If this worksheet has been removed from the DOS/VS COBOL PLM, 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.</p>		

COBOL Abstract Worksheet (Side B)

Search Item	Search Argument	How to Find Argument
1. Component ID: 5746-CB-xxx, where xxx is the program product version level.	1. 5746-CB-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>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. = MSGILAXxxxx bad output = INCORROUT	3.	
4. COBOL verb statement being executed:	4.	<p>A. Determine 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>B. Determine the length of the compiled CSECT from one of the following areas:</p> <ul style="list-style-type: none"> <li>• linkage editor map</li> <li>• extent list</li> <li>• compiler generated Memory Map</li> </ul> <p>C. Use the PSW AT ENTRY TO ABEND to determine the address of the failing instruction.</p> <p>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, SYMDUMP)	6.	
<p><b>Note 1.</b> (a) Calculate the displacement of the failing instruction from the CSECT entry point. (b) Find this displacement in the Memory Map assembler listing. (c) The statement number related to the generated code is printed on the left side of the assembler listing. In some cases, the COBOL verb is also printed in the listing.</p> <p><b>Note 2.</b> (a) Trace back from failing CSECT to exit from COBOL CSECT. (b) Use the linkage editor map, the CDE, and extent list or the PSW in the PRB in conjunction with the SAVE AREA TRACE to find the name of the failing CSECT and the entry point for the CSECT. (c) If the return address saved by the failing CSECT is within the scope of the compiled CSECT, proceed as in Note 1. If not, continue the traceback.</p>		

APPENDIX A. TABLE AND DICTIONARY HANDLING

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 area used by phases 11, 21, 22, 25 and 30, is manipulated by a set of routines called ACCESS routines. ACCESS routines are loaded into storage as part of these phases.

The chapter on phase 01 explains how a region for tables and the dictionary is obtained. Figure 72 shows the arrangement of tables and the dictionary.

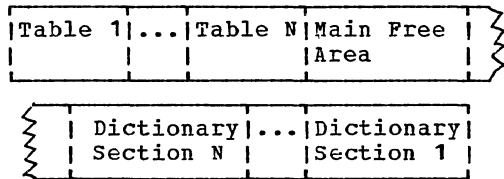


Figure 72. Arrangement of Tables and Dictionary Sections in a Contiguous Region

ACCESS DICTIONARY HANDLING ROUTINES

ACCESS routines enter and retrieve dictionary entries. They are assembled by means of a macro instruction, with phases 11, 21, 22, 25, and 30 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 11, 21, and 22 make dictionary entries for data-names and procedure-names in the order in which the names are defined in the source program. (Phase 11 makes entries for procedure-names. Phases 21 and 22 make 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 11, 21, and 22 do not use LOCNXT.

Phase 30 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 30 does not use 'enter' or 'get' ACCESS routines.

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. For each dictionary section, table DICOT has an entry which gives the starting address of that section.

The ACCESS routines use table HASH 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 in order 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 table HASH, and an indication is made that there were duplicate hash values. 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 looks at the name in the indicated entry. If the names match, this is the correct entry, unless duplicate names were defined. Then the ACCESS routine looks at the entry with the section and displacement specified by the dictionary pointer of the last entry. This process continues until the names of all entries with duplicate hash values have been compared. At that time, the ACCESS routine has found a single unique name, a duplicately defined name, or no name at all. It issues an error indication 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.

AREA FOR THE DICTIONARY

Area for a new dictionary section can be obtained from the free area between the tables and the dictionary.

When the outstanding available area is exhausted, dictionary sections are written on disk and the area is reused. Table DICOT 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 (spilled) on disk. For further information, see subroutine MOVDIC in the section "Table Handling with TAMER" in this appendix.

When control passes from a smaller to a larger phase, any tables in danger of being overlaid must be moved to upper storage.

The ACCESS routines do not restore registers 0 and 1 on exit.

In the following descriptions of the ACCESS routines, the format of the BCD name pointed to is:

No. of Bits:	2	6	n*8
Contents:	00	n	name

where n is the number of characters in the name. The entry starts on a word boundary and is a multiple of four bytes. Padding is done with zeros starting with the low-order bytes.

INITIALIZATION OF ACCESS ROUTINES

In order to use the ACCESS routines, the phases must call routine INTACC to initialize them. INTACC primes the DICOT table and performs other initialization functions. The call to INTACC must follow the call to TAMEIN, the routine that initializes the TAMER. 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 an 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.

ENTNAM (Enter Attributes Given Name)

Given the address of a BCD name (procedure-name or data-name), routine ENTNAM makes a dictionary entry for it. It places its section and displacement in table HASH and also in register 1, for use by the calling phase as a dictionary pointer. The calling sequence is:

```
L      1,=A(parameter)
L      15,=A(ENTNAM)
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 BCD Name	Count of Attributes	Address of Attributes

where code is 0 for elementary items and paragraph-names and 4 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
```

where parameter has the following format starting on a fullword boundary:

No. of Bytes	1	3	1	3	1	3
Contents	16	Ad- dress of BCD Name	Count of Attri- butes	Ad- dress of Attri- butes	0	Dic- tionary Pointer

GETPTR (Get Pointer)

Given a BCD 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
Contents	Code	Address of BCD name	Count of Attributes

where code is 8 for elementary items and paragraph-names and 12 for group items and section-names.

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 a BCD 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 zero. If the entry is not found, the contents of registers 2 and 3 are meaningless. Register 15 contains a 4 if the name was not found and an 8 if the name was duplicatedly defined. 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 BCD 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 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 BCD name of the next entry. The calling sequence is:

```
L      1,=A (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 BCD 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. It places the dictionary pointer of the data-name in register 3. It places the delimiter pointer in register 1. It sets register 15 to zero.

If an error is detected, one of the following codes is placed in register 15.

1. If the unique name located was a paragraph-name or elementary item name, register 15 is set to 12. Registers 2 and 3 are set as above.
2. If the name is not found, register 15 is set to 4. Registers 2 and 3 contain meaningless information.
3. If the name is duplicately defined, register 15 is set to 8. Registers 2 and 3 contain meaningless information.

The calling sequence 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 BCD Name

LATACP (Locate Attributes Using ACCESS Pointer)

Given the BCD 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. Register 15 is set to zero. (This routine is used to locate qualified names. It limits the search of the dictionary.)

If an error is detected, a code is placed in register 15: the code is 4 if the name was not found, 8 if the name was duplicately defined, and 12 if the last entry referred to was an elementary item. Registers 2 and 3 contain meaningless information. The calling sequence 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	0	Address of BCD Name



**Note:** There must have been no call to a TAMER routine intervening between routine LATACP and the last call to an ACCESS routine.

LATGRP (Locate Attributes Given Group Pointer)

Given the BCD 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 zero.

If an error occurs, it places a code in register 15: the code is 4 if the name was not found, 8 if the name was duplicately defined, and 12 if the given dictionary pointer pointed to an elementary item or paragraph-name. Registers 2 and 3 contain meaningless information. The calling sequence 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 BCD Name	0	Group or Section Dictionary Pointer

TABLE HANDLING WITH TAMER

TAMER (Table Area Management Executive Routines) resides permanently in 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.

TIB (Table Information Block)

For each table, there is a TIB in a fixed location. 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 TAMB (see below). Each TIB has the following format:

No. of Bytes	1	3	2	2
Contents	Entry Length	TAMB Address	Table Length	Growth Factor

- Entry Length**  
number of bytes in a table entry.
- TAMB Address**  
address of the TAMB 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.

TAMB (Table Area Management Map)

For each table there is a TAMB in a variable location within a fixed block (the TAMB block). Each TAMB points to a table and to the TIB for the table. The format of a TAMB 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:

Code	Meaning
01	Indicates that the table has been released so that its area is available as free area.
02	Indicates that the table has been set static so that 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.

Note: Since the N1 field is two bytes long, the maximum length of any table handled by TAMER is limited both by the amount of storage available to the programmer and by the maximum size that can be described in two bytes, or 32K-1 (32,767) bytes. If either limit is exceeded, compilation will terminate and an error message will be issued. This limitation does not apply to the dictionary, which uses SYS001 as a spill file, or to the RLDTBL table in phase 60, or to the CNTLTBL, DATATBL, and OFLOTBL tables in phase 61 which are not built by TAMER and hence are limited only by the size of storage available.

MASTAM (Master TAMM Table)

The MASTAM contains the characteristics of the area in storage assigned to TAMER. The MASTAM has the following format:

Beginning of area
Length of area
First free byte not used so far
Length of free area left over
First TAMM used
Next TAMM to be used
Number of dictionary sections within the area

HOW SPACE IS ASSIGNED

At the beginning of compilation, the space between the end of the largest phase and the beginning of the buffer area is defined as the main area of storage for TAMER. The MASTAM is set up for the area. If more space is needed later on, COBOL space is allocated (COBOL space is the difference in length between the longest phase and the current phase).

If the next phase to process is larger than the current phase, the tables which are to be passed between phases are packed and moved so that none of them are overlaid by the current phase.

Within the TAMER area, the tables start in lowest storage and the dictionary starts in highest storage. The TAMMs are assigned contiguously within the TAMM block, and their order reflects the storage order of the tables to which they point.

TAMEIN Routine

TAMEIN is the TAMER initialization routine. It is called during phase 00 before linking to phase 30. It is also called by Phase 01 (for BASIS or COPY) or Phase 10. Its operations are as follows:

Phase 01 Called before any other TAMER or 10: routine. Sets up the first MASTAM for the table 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 00: Before phase 30, if the dictionary was not spilled (that

is, if no dictionary sections were written on an external device), no action is taken. If the dictionary 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.

where PARAM 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

The calling sequence for TAMEIN is:

```
L      15,=A(TAMEIN)
BALR   14,15
```

TBGETSPC Routine

Routine TBGETSPC is called by routine PRIME to obtain COBOL space. The area between the current starting table address and the end of the current phase is made available to TAMER routines.

PRIME Routine

Routine PRIME allocates space 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 space of the main area.
2. A check is made for the required area in the space made available because of the release of tables by TAMER -- called freed area.
3. An attempt is made to pack the tables (eliminating the free bytes between the primed and static tables) to make the main area larger.
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 on an external device). 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 TAMB 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 TAMB is not created and the calling routine takes whatever action is necessary.

The calling sequence for PRIME is:

```
L      1,=A(PARAM)
L      15,=A(PRIME)
BALR   14,15
```

MOVDIC Routine

Routine MOVDIC reads back into storage a dictionary section which has been spilled. First, MOVDIC calls the PRIME routine to make space available for the section and, then, reads the section back into the space made available.

The calling sequence for MOVDIC is:

```
L      15,=A(MOVDIC)
L      3,=A(DICOT table entry)
BALR   14,15
```

DICSPC Routine

Routine DICSPC is called only by an ACCESS routine, and requests space for a dictionary section. The space is provided by an internal call to routine 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 sets a table static. This means that no new entries will be made in the table during the rest of the phase. It sets the TAMB 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 free area. It sets the TMM for the table to released format, as follows:

No. of Bytes	1	3	4
Contents	01	Table Address	Released Bytes

The table address field is set to zero. Both the TMM 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 TMM for the table and returns to the phase the starting address (in register 2) and the displacement (in register 3) of the entry.

If the area allocated to the table will not hold the entry, routine INSERT calls the PRIME routine to obtain additional space.

The phases call routine 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. Alternatively, the current entry length can be placed in register 0 and the entry length field of the TIB set to 0. When this is done, the number of bytes currently occupied by table entries is returned to the phase in register 3.

TAMEOP Routine

Routine TAMEOP is called at the end of every phase to reset TAMER switches. It also handles the passing of tables between phases.

The calling sequence for TAMEOP is:

```
L      15,=A(TAMEOP)
BALR   14,15
```

TBSPILL Routine

Routine TBSPILL is called by routine PRIME, checks for the last dictionary section (the section in highest storage), and calls routine TBWRITE to spill it in order 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, the TBSPILL routine 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 an external device. the TBWRITE routine uses the DICOT table to check and indicate the status of dictionary sections (see "ACCESS Dictionary Handling Routines" in this appendix).

If a section has never been spilled, routine TBWRITE spills it by issuing the WRITE macro instruction. If a section has been spilled before, but has been changed since then, routine TBWRITE issues the POINTW macro instruction followed by WRITE to put the fresh copy on the external device. If a section has been spilled and has not been changed since then (that is, an exact copy already exists on an external device), it is not spilled again.

TBREADIC Routine

Routine TBREADIC is called by routine MOVDIC to read a dictionary section back into storage.

GETALL Routine

Routine GETALL is called by the Lister phases and phases 60 and 61. Its function is to provide space for a table which may be in excess of 32K bytes, the normal maximum size. This routine requests all available table space in a contiguous area. This request may be made only when all current tables have been set static. The tables are packed, and GETALL passes the

starting address and length of the remaining table area back to the calling phase in registers 0 and 1, respectively.

All subsequent use of that area is handled internally by the phase which called routine GETALL, since a call to the TABREL routine is the only TAMEER call which may legitimately follow a call to GETALL in the same phase. At the end of the phase which called routine GETALL, the area then becomes available for normal phase 00 table-handling procedures.

In the event that additional space is needed after some tables have been released, a second call to the GETALL routine is issued. TAMEER then packs the tables again and uses the additional space obtained to expand the original GETALL area.

APPENDIX B: OBJECT MODULE

This appendix describes the object module produced by the compiler for input to the linkage editor. The fields in the object module are shown in Figure 73 as they appear after linkage editing. The fields are discussed in separate sections as they appear in this figure.

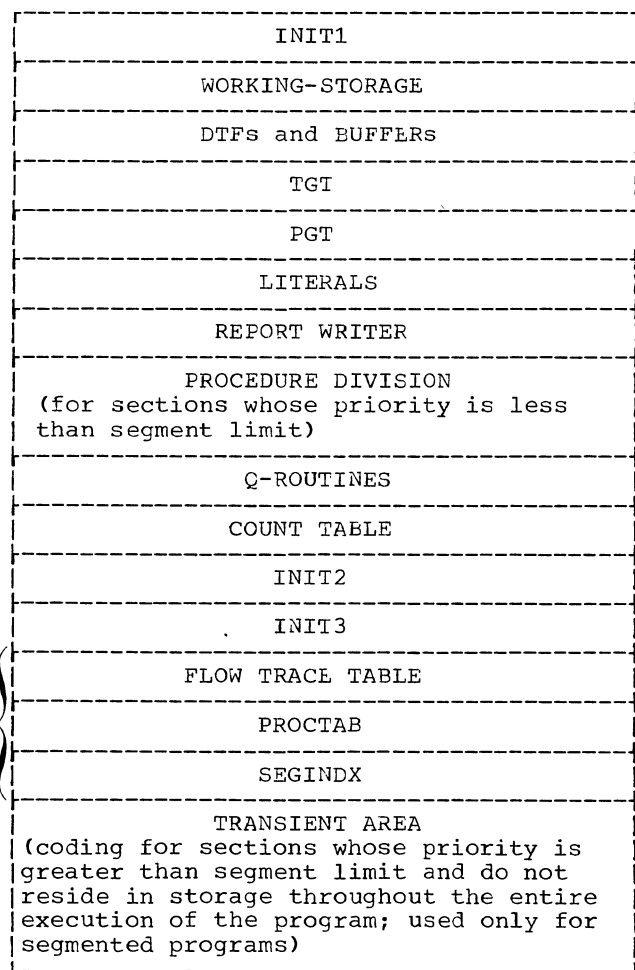


Figure 73. Storage Map of Object Module Fields

INITIALIZATION 1 ROUTINE (INIT1)

The Initialization 1 routine begins at relative location zero and is constant in length for every compilation. It performs the following:

1. Saves the calling program registers and the pointer to its Task Global Table or save area.
2. Sets up 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 if it is a subprogram, or to INIT3 if it is a main program.
4. Passes the address of the transient area to INIT2 in a segmented program.
5. Contains a save area so that the program can be reset to the state it was left in upon re-entry.
6. Branches to ILBVOC0 COBOL Library Subroutine to initialize for input/output verbs with VSAM files.

The code generated by INIT1 is shown in Figure 74. The same block of code at the end of the object module differs in that the relative addressing of the object module is here shown in a symbolic manner.

WORKING-STORAGE

This area contains reserved storage for the Working-Storage Section of the Data Division. The data items for which VALUE clauses were specified have been initialized.

DTF'S AND BUFFERS

This area contains data areas from the Data Division, including SDTFs, DTFs, record areas, and buffers. Buffers for the SORT SDs may be intermixed with DTFs.

TASK GLOBAL TABLE (TGT)

The TGT is used to record and save information needed during the execution of the object program. It consists of a fixed and a variable portion. In the fixed portion are those fields for which space is allocated for every object program. The variable portion is tailored to a particular object program. The fields in the TGT are shown in Figure 75 and are described in alphabetical order following the table.

Hexadecimal location					
0000	INIT1	BALR	15,0		
		BCR	0,0		
		STM	0,14,SAVEP1	Save Registers	
		BC	15,TRAN000		
000C	SAVEP1	DS	30F	Register Save areas for entry and exit	
	TRAN000	L	12,ADCON003	Load with address of PGT	
		L	14,=V(ILBDMNS0)	Get address of byte MAIN.SUE SWITCH	
		L	13,ADCON004	Load with address of TGT	
		CLI	0(14),0		
		BC	7,TRAN001	Not a main program	
		OI	SWITCH,X'10'	Set main program switch (Displacement is 48 (hex) from start of TGT)	
		MVI	0(14),X'FF'		
		BC	15,TRAN002		
00A4	TRAN001	LM	12,14,SAVEP1+48	For Subroutine; again save registers to comply with standard linkage convention	
		STM	14,12,12(13)		
		LR	5,13		
00AE	TRAN002	LM	9,15,ADCON000		
		TM	SWITCH,X'10'		
		BCR	1,9	For main bypass saving of 13	
		ECR	15,15		
		BCR	0,0		
00BC	ADCON000	DC	A(INIT3)		
	ADCON001	DC	A(INIT1)		
	ADCON002	DC	A(INIT1)		
		or			
		DC	A(SEGMT)	If program is segmented	
	ADCON003	DC	A(PGT)		
	ADCON004	DC	A(TGT)		
	ADCON005	DC	A(START)		
	ADCON006	DC	A(INIT2)		
00D8		DC	X'C3D6C2C6'	COBF compiling program-name	
		or			
		DC	X'CBD6C2D6'	COBOL compiling program-name if compiled with OPT	
00DC		DC	X'F0F0F0F0'	Version number, level of modification number	
00E0		DC	X'NNNNNNNN'	Program-id name	
		DC	A(BLL2)	Address of BLL2 if program has Linkage Section	
		or			
		DC	X'0000'	If program has no Linkage Section	
		DC	X'F9F961F9F961F9F9'	Date of compilation	
		DC	X'F1F242F5F942F5F9'	Time of compilation start	

Figure 74. INIT1 Coding

Relative Address (Hex)	(Dec)	Field
0	0	SAVE AREA
48	72	SWITCH
4C	76	TALLY
50	80	SORT SAVE
54	84	ENTRY-SAVE
58	88	SORT CORE SIZE
5C	92	NSTD-REELS
5E	94	SORT RET
60	96	WORKING CELLS
190	400	SORT FILE SIZE
194	404	SORT MODE SIZE
198	408	PGT-VN TBL
19C	412	TGT-VN TBL
1A0	416	SORTAB ADDRESS
1A4	420	LENGTH OF VN TBL
1A6	422	LENGTH OF SORTAB
1A8	424	PGM ID
1B0	432	A(INIT1)
1B4	436	UPSI SWITCHES
1BC	444	DEBUG TABLE PTR
1C0	448	CURRENT PRIORITY
1C1	449	TA LENGTH
1C4	452	PROCEDURE BLOCK1 PTR
1C8	456	Unused
1CC	460	COUNT TABLE ADDRESS
1D0	464	VSAM SAVE AREA
1D4	468	Unused
1DC	476	COUNT CHAIN ADDRESS
1E0	480	Reserved
1F4	500	OVERFLOW CELLS
		BL CELLS
		DTFADR CELLS
		FIB
		TEMP STORAGE
		TEMP STORAGE-2
		TEMP STORAGE-3
		TEMP STORAGE-4
		BLL CELLS
		VLC CELLS
		SBL CELLS
		INDEX CELLS
		SUBADR CELLS
		ONCTL CELLS
		PFMCTL CELLS
		PFMSAV CELLS
		VN CELLS
		SAVE AREA = 2
		XSASW CELLS
		XSA CELLS
		PARAM CELLS
		RPTSAV AREA
		CHECKPT CTR
		IOPTR CELLS
		DEBUG TABLE

Note: The portion of the TGT following the OVERFLOW CELLS consists of variable-length fields.

Figure 75. TGT Fields

BL CELLS

Base locators. These are fullwords containing the addresses of data areas in object module field DATA. Phases 22 and 21 assign a base locator to each block of 4096 bytes in the Working-Storage Section and to each file in the File Section.

BLL CELLS

Base locators for Linkage Section. These are fullwords containing the addresses of areas passed as a result of ENTRY statements, label records, or the GIVING option in USE...ERROR clauses.

CHECKPT CTR

Four-byte counters used to count the numbers of records processed for files for which checkpoints are to be taken.

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.

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.

CURRENT PRIORITY

Priority of the segment currently in the transient area. If the STATE or SYMDMP option is specified, the segmentation subroutine (ILBDSEM0) inserts the priority in this cell. It is initialized to 0 by phase 65.

DEBUG TABLE

This table is used by the SYMDMP (ILBDMP10-ILBDMP25), FLOW (ILBDFLW0), and STATE (ILBDSTN0) COBOL library subroutines. It is built by phase 65; the format depends on the options specified. (See Figure 76: Debug Table Formats)

DEBUG TABLE PTR

A fullword containing the displacement from the beginning of the TGT to the DEBUG TABLE field if SYMDMP, STATE, or FLOW option was specified.



DTFADR CELLS

DTF address. There is one fullword containing the address in the object module of each DTF used by the object module.

ENTRY-SAVE

A 4-byte cell used to save the entry point of the object program during INIT2 and INIT3 execution.

FIB

File Information Block addresses. There is one fullword cell pointing to the address of the FIB for each VSAM file. At initialization time, it is changed to contain the address of the FIB for that file.

INDEX CELLS

Index-name cells for each index name in the program.



Options	Length	Byte	Contents
If FLOW is specified:	4	0 1-3	Number of traces requested. Address of table area to be used by FLOW subroutine.
If FLOW and STATE are specified:	24	0 1-3 4-7 8-11 12-15 16-19 20-23	Number of traces requested. Address of table area to be used by FLOW subroutine. Address of beginning of Q-routines, or if none, INIT2. Size of Declaratives Section within Procedure Division. Address of beginning of PROCTAB in object module. Address of beginning of SEGINDX in object module. Address of end of SEGINDX.
If FLOW and SYMDMP are specified:	10	0-7 8-9	The same as shown for bytes 0-7 for FLOW and STATE. Hashed compilation indicator. See "PROGSUM Table" in "Section 5. Data Areas."
If STATE is specified:	20	0-19	The same as shown for bytes 4-23 for FLOW and STATE.
If SYMDMP only is specified:	10	0-3 4-7 8-9	Unused. Address of the beginning of Q-routines or, if none, INIT2. Hashed compilation indicator. See "PROGSUM Table" in "Section 5. Data Areas."

Figure 76. Debug Table Formats

A(INIT1)

Address of INIT1 for GOBACK, STOP RUN, and EXIT PROGRAM instructions, and for segmented coding.

IOPTR CELLS

Four-byte cells used when SAME RECORD AREA clause is specified.

LENGTH OF SORTAB

A halfword, reserved but not presently being used.

LENGTH OF VN TBL

Two-byte field containing the length of the VNs for the independent segment.

NSTD-REELS

One-byte field for nonstandard labeled reels.

ONCTL CELLS

ON control counters. These are fullwords that control the execution of ON statements. They are initialized to zero. There is one ONCTL for each ON statement in the program.

OVERFLOW CELLS

One pointer to each 4096-byte section after the first in the TGT, if necessary.

PARAM CELLS

Parameter area, consisting of fullwords. It contains parameter lists for certain verb expansions in the source program. The size of the parameter area is equal to the largest number of words required for any one verb's uses.

PFMCTL CELLS

Control counters for PERFORM statements. PFMCTL cells are fullwords that control the execution of PERFORM n TIMES statements. There is one PFMCTL cell for each PERFORM n TIMES statement in the program.

PFMSAV CELLS

Saved locations for PERFORM and DEBUG statements. These are fullwords that contain addresses which enable the source program logic to execute in-line a paragraph that is also the object of a PERFORM statement. There is one PFMSAV cell for each PERFORM statement in the in-line procedure. In addition, there is one full word to hold the contents of register 14 upon entering a debug packet. Debug packets are called by BALR 14,15.

PGM ID

An 8-character root segment name. It is used by the COBOL library subroutine ILBDSEM0 to generate names of nonresident segments.

PGT-VN TBL  
A fullword pointer to the VN cells in the PGT belonging to independent segment.

PROCEDURE BLOCK1 PTR  
A fullword containing the address of the first PROCEDURE BLOCK CELL in the PGT.

RPTSAV AREA  
Six words used to save branch addresses during the execution of Report Writer routines, if necessary.

SAVE AREA  
Save area for the program.

SAVE AREA=2  
Save area pointer provided for label- and error- processing declaratives.

SBL CELLS  
Secondary base locators. These are fullwords, set by the execution of a Q-routine, each containing the address of a field that has a variable location because it follows a variable-length field.

SORT CORE SIZE  
Four bytes containing the SORT-CORE-SIZE special register as specified in the source program.

SORT FILE SIZE  
Four bytes containing the SORT-FILE-SIZE special register as specified in the source program.

SORT MODE SIZE  
Four bytes containing the SORT-MODE SIZE special register as specified in the source program.

SORT RET  
A halfword containing the return code from a SORT or MERGE operation.

SORT SAVE  
A fullword used to save a GN cell during the execution of a SORT or MERGE RETURN statement.

SORTAB ADDRESS  
A fullword, reserved but not currently used.

SUBADR CELLS  
Subscript addresses. These are fullwords each containing the address calculated from a subscripted reference. They are filled in at execution time, not written as text.

SWITCH  
A fullword switch. Only the following bits are used.

Bit	Meaning
0	Set to 1 if SIZE ERROR for ADD and SUBTRACT CORRESPONDING
1	Set to 1 for TRACE READY; set to 0 for RESET TRACE
2	Set to 1 when initialization is performed for the program
3	Set to 1 if the program is a main program; set to 0 if it is a subprogram
4	Used for SYMDMP. It is set to 1 by phase 65 if the SYMDMP option is in effect for the program. This bit is tested by the COBOL library Debug Control subroutine (ILBDDBG0).
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 COBOL library Debug Control subroutine (ILBDDBG0).
6	Used for STATE. It is set to 1 by phase 65 if the statement number option is in effect for the program. This bit is tested by the COBOL library Debug Control subroutine (ILBDDBG0).
7	Used for OPT. Set to 1 by phase 62 if optimization has been requested for this program. This bit is tested by the COBOL library SORT subroutine (ILBDSRT0).
8	Used by IBM DOS Subset American National Standard COBOL compiler. It is set to 1 in a subset compilation to differentiate between full and subset COBOL object programs.
10	Used for segmentation. Set to 1 by the IBM DOS Subset American National Standard COBOL compiler if there is segmentation in this program.
11	Used for SORT. Set to 0 for V3 or to 1 for VS.
15	Set to 1 if OCCURS...DEPENDING ON maximum length is to be calculated. Set to 0 if OCCURS...DEPENDING ON actual length is to be calculated.
20	COUNT switch
23	Used by ILBDSRT0 and ILBDCKP0. Set to 1 on entry to ILBDSRT0; set to 0 if checkpoint restart occurs.
24-31	DECIMAL-POINT IS COMMA

**1A LENGTH**  
A halfword initialized by phase 60 to the length of the largest segment with a nonzero priority.

**TALLY**  
A fullword for source program reference to the TALLY special register.

**TEMP STORAGE**  
Variable-length cells analogous to WORKING CELLS but used by arithmetic procedure instructions. Each must start on a doubleword boundary. They are allocated by phase 50 in blocks of 8 bytes.

**TEMP STORAGE-2**  
Variable-length cells analogous to WORKING CELLS but used by nonarithmetic procedure instructions.

**TEMP STORAGE-3**  
These cells are the same as TEMPORARY STORAGE-2, but used for SYNCHRONIZED clause handling. They start on a doubleword boundary.

**TEMP STORAGE-4**  
These cells are the same as TEMPORARY STORAGE-2, but used for table handling. They start on a doubleword boundary.

**TGT-VN TBL**  
A fullword pointer to VN cells in the TGT belonging to the independent segment.

**UPSI SWITCHES**  
User Program Switch Indicators. This is an 8-byte field containing switches that can be tested by a problem program.

**VLC CELLS**  
Variable-length cells. These are halfwords, set by the execution of a Q-routine, each containing the current length of a variable-length field. There is one VLC for each variable-length field.

**VN CELLS**  
Variable procedure-names. There is one fullword VN containing the current address of each procedure-name in the object module field PROCEDURE that is altered by an ALTER statement or is the procedure-name of a paragraph that follows a paragraph (or a series of paragraphs) which is in the range of a PERFORM statement.

**VSAM SAVE AREA**  
Fullword pointer to the save area used

by the VSAM COBOL Library Subroutines (ILBDVOC0 and ILBDVIO0).

**WORKING CELLS**  
Variable-length cells used by COBOL library subroutines called by the generated code. The total length of the field is 304 bytes.

**XSA CELLS**  
EXHIBIT saved areas. These are variable in length and are referred to in the coding generated for an EXHIBIT statement with a CHANGED option. There is one XSA for each operand to be exhibited with a CHANGED option in the in-line procedure. These areas are also used for SORT and RELEASE verbs.

**XSASW CELLS**  
One-byte EXHIBIT switches. These are used as first-time switches for the coding generated for the EXHIBIT CHANGED statement. They are also used in miscellaneous cases of SORT statements and special cases of ON statements.

PROGRAM GLOBAL TABLE (PGT)

The Program Global Table contains address constants and literals referred to by procedure instructions. The fields are shown in Figure 77.

DEBUG LINKAGE AREA
COUNT LINKAGE AREA
OVERFLOW CELLS
VIRTUAL CELLS
PROCEDURE NAME CELLS
GENERATED NAME CELLS
SUBDTF ADDRESS CELLS
VNI CELLS
LITERALS
DISPLAY LITERALS
PROCEDURE BLOCK CELLS

Figure 77. PGT Fields

**DEBUG LINKAGE AREA**  
Eight-byte area which contains the linkage for dynamic dumps. The area contains the following code:

```
L  11,=V(ILBDDBG5)
BR  11
(2 slack bytes)
```

If the SYMDMP option is not specified, this 8-byte area does not exist.

**COUNT LINKAGE AREA**  
Eight-byte area which contains the

linkage to the COUNT routine. If the COUNT option is not specified, this 8-byte area does not exist. The area contains the following code:

```
L    15,=V(ILBDC710)
BR    15
DC    1H'0'
```

**OVERFLOW CELLS**

One pointer to each 4096-byte section after the first in the Program Global Table.

**VIRTUAL CELLS**

A virtual is a fullword containing the address of each unique external procedure (the result of an ESD and RLD in the object module). It is required because of a CALL statement to the external procedure or a branch to a COBOL library subroutine.

**PROCEDURE NAME CELLS**

Source procedure-names. Each PN cell is a fullword containing the address of an instruction. It corresponds to the procedure-name in the source program, except that a source program procedure-name which is not referenced is not assigned a PN. When OPT has been specified on the CBL card, only those PNs associated with ALTER and DECLARATIVES references receive PN cells.

**GENERATED NAME CELLS**

Compiler generated procedure-names. Each GN cell is a fullword containing the address of an instruction. The compiler assigns a GN wherever necessary in order to branch around generated code statements. When OPT has been specified on the CBL card, only those GNs associated with AT END and INVALID KEY receive GN cells.

**SUBDTF ADDRESS CELLS**

SDTF addresses. Each SDTFADR is a fullword containing the address of an SDTF in the "DTFs and BUFFERS" field of the Object Module. There is one SDTFADR cell for each SDTF generated by the compiler.

**VNI CELLS**

Variable procedure-name initialization cells. There is one VN for each variable procedure-name. It contains the initial address and is used to initialize the VN locations in the Task Global Table.

**LITERALS**

Literals that are referred to by instructions. Duplicate literals are deleted during phase 60 or 62 processing. The literals are variable in length.

**DISPLAY LITERALS**

Literals that are referred to by calling sequences, rather than by instructions. Duplicate literals are deleted during phase 60 or 62 processing. The literals are variable in length.

**PROCEDURE BLOCK CELLS**

Each cell is a fullword containing the address of a Procedure Block. The compiler assigns these cells when OPT is specified.

REPORT WRITER

See "Appendix C: Report Writer Subprogram."

PROCEDURE DIVISION

This area contains the generated code for the Procedure Division of the source program. If the program is segmented, the root segment is loaded here (see "Transient Area" in this appendix).

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

```
INIT2    ST    13,008(0,5)      For subroutine only; chain
         ST    5,004(0,13)     save area of called and calling programs.
         L     2,=V(ILBDMNS0)  In case program was entered
         CLI  0(2),X'00'      from a secondary entry point,
         BCR  7,9             check whether it is a main
         MVI  0(2),X'FF'      or a subprogram.
         OI   SWITCH,X'10'
```

Figure 78. INIT2 Coding

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).

The Q-routine only 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 if the OCCURS...DEPENDING ON area becomes longer, the information that had immediately followed the area before it changed is now no longer accessible. The pointer to it in the SBL has been moved. The source programmer can avoid a loss of data by moving it 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 allocated enough space for the maximum number of occurrences.

The generating of Q-routines is discussed under "Q-routine Generation" in the Phase 22 chapter.

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

-----  
 \*This field is not present if byte 0 contains 00.

- \*7                    non-executable verbs.  
                       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 routine is generated by phase 60 or 64 and performs the following operations:

1. Stores the address of the Task Global Table for this program (pointed to by register 13) into the Save Area of the Task Global Table of the calling program.
2. Stores the location of the Save Area of the Task Global Table of the calling program into the Save Area of this program's Task Global Table.
3. Determines if this module is a main program or a subprogram in the run unit and sets a switch accordingly.

The code generated by INIT2 is shown in Figure 78. Symbolic addressing has been substituted in some instructions (see the INIT1 coding above).

INITIALIZATION 3 ROUTINE (INIT3)

The Initialization 3 routine is executed whenever the program is entered. It is generated by phase 60 or 64 and immediately checks the SWITCH field in the TGT to determine whether this is the first time the module was entered. If so, it performs the following operations:

1. Initialize the VN locations in the Task Global Table from the associated VN locations in the Program Global Table.
2. Relocate each address constant in the Task Global Table and the Program Global Table to its absolute location. (Before the execution of this routine, the addresses are relative to the beginning of the program.) The relocation of the address constants for this program (or for only the root segment in a segmented program) is done by adding the absolute location of the first instruction in the program (Initialization 1 routine), which is in general register 11, to the relative addresses. If the high

byte of an ADCON is not zero, the address of the transient area (contained in register 10) is used as a relocation factor. If OPT is specified, the addresses contained in the PROCEDURE BLOCK CELLS in the PGT are processed as follows: If the program is not segmented, RLD-text is generated for the PROCEDURE BLOCK CELLS. If the program is segmented, all PROCEDURE BLOCK CELLS are relocated by INIT3.

3. Load and BALR 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. Load permanent base registers for BLs assigned to the program and, if OPT is specified, for BLs, BLLs and OVERFLOW CELLS.
5. Branch either to the first executable instruction of the object program, or to the instruction following the BALR to the Initialization 2 routine in the coding generated for an ENTRY statement.
6. If the module was entered previously, INIT3 resets the program to the state it was left in the last time it was exited (from the register save area in INIT1).

The code generated by INIT3 is shown in Figure 79. Symbolic addressing has been substituted in some instructions (see the INIT1 coding above).

FLOW TRACE TABLE

The Flow trace table is used by the COBOL library subroutine (ILBDFLW0), if FLOW has been specified.

The compiler only allocates the area for the table, the size of which is dependent on the number (n[n]) of traces requested; the COBOL library subroutine (ILBDFLW0) makes the entries in the table.

PROCTAB TABLE (PROCTAB)

The PROCTAB table is used by the COBOL library subroutine (ILBDSTN0). It contains entries for all the card numbers and verb numbers in the COBOL program. The format of each entry is:

<u>Byte</u>	<u>Contents</u>
0-2	Card number and verb number. The verb number is contained in the last 4 bits of byte 2.
3-4	Displacement of the verb within the program fragment.

SEGINDX TABLE (SEGINDX)

The SEGINDX table is used by the COBOL library subroutine (ILBDSTN0). It contains an entry for each fragment of the program; these entries are written in ascending order of priority in the object module. The format of each entry is:

<u>Byte</u>	<u>Contents</u>
0	Priority.
1-3	Program fragment address.
4-6	Displacement from start of PROCTAB table of first PROCTAB entry for this program fragment.
7-9	Displacement from start of PROCTAB table of last PROCTAB entry for this program fragment.



```

INIT3  ST      14,0054(0,13)  Save entry point to go to after INIT3
                               (See Note)
      BALR    15,0
      TM      SWITCH,X'20'
      BC      14,TRAN003      Register 11 used as base
      L       0,SAVEP1+48
      LM      2,13,SAVEP1+50  Restore registers to what they were on last exit
      L       14,0054(0,13)
      BCR     15,14
TRAN003 OI      SWITCH,X'20'
      LA      6,4

      Relocate OVERFLOW, PN, GN, SUBDTF, and VNI cells of the PGT.

      If OPT is specified and the program is segmented, relocate all PROCEDURE BLOCK
      CELLS in the PGT.

      Relocate OVERFLOW, BL, and DTFADR cells of the TGT.

      Call Q-routine for OCCURS...DEPENDING ON fields which have their object in
      Working-Storage.

      Load base locators.

      Call initialization of checkpoint routine.

      L       14,0054(13)
      BCR     15,14
    .

```

---

Note: If the SYMDMP, FLOW, STATE, or COUNT option is in effect, the following instructions are interposed between the first instruction (ST 14,0054,(0,13)) and the second instruction (BALR 15,0) of INIT3 coding.

```

      L       15,=V(ILBDDBG0)
      BALR    14,15

```

In addition, if COUNT is in effect, the following instruction is added after the BALR:

```

      DC      H 'number of verbs blocks'

```

Figure 79. INIT3 Coding

TRANSIENT AREA (SEGMENTED PROGRAM)

This area contains object modules of program segments of a higher priority than the segment limit (49 by default). The segments are compiled in ascending order of priority number, and the root segment is compiled last. If OPT is specified, Phase

64 compiles the root segment first and then the other segments. (The root segment is stored in the Procedure Division area of the object module, as explained above.) Assuming the name of the program is SORTEST, and the segments are of priorities 51, 52, and 0, the object module produced by the compiler has the following format:



ROOT SEGMENT      PHASE      SORTEST,ROOT  
                    (Object deck for the root  
                    segment)

INDEPENDENT      PHASE      SORTES51,\*  
SEGMENT           (Object deck for segment of  
                    priority 51)

INDEPENDENT      PHASE      SORTES52,SORTES51  
SEGMENT           (Object deck for segment of  
                    priority 52)  
                    ENTRY      SORTEST

A root segment consists of the Data Division specified areas, including Report Writer routines, O-routines, Global Tables, INIT1, INIT2, INIT3, and Procedure Division coding for sections of a priority less than the segment limit.

Each nonresident segment will be loaded into the transient area when necessary. The transient area is large enough to accommodate the largest segment with a nonzero priority.

If no root segment is specified, one is generated by the compiler.

APPENDIX C: REPORT WRITER SUBPROGRAM

This appendix 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 "Phase 12" chapter.

If OPT 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 the code generated for the other parts of the Procedure Division is optimized. 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. Each routine refers to the compiler-generated card number of its respective RD. This is reflected when LISTX is in effect. Figure 80 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 below, 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

This routine causes the first headings to be printed by calling on the RPH-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

This 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 LAST-ROLL, 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

This 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".

PARAMETRIC ROUTINES

Parametric routines are generally fixed in structure but vary according to the data obtained from the source (01-49 level) 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

This 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 which 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

This 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

This 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

This 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

This 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

This 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 (i.e., the value prior to the control break).

#### INT-ROUT Routine

This 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

This routine determines the line spacing for absolute lines. The ALS-ROUT routine is generated only if the RD entry contains a PAGE LIMIT clause.

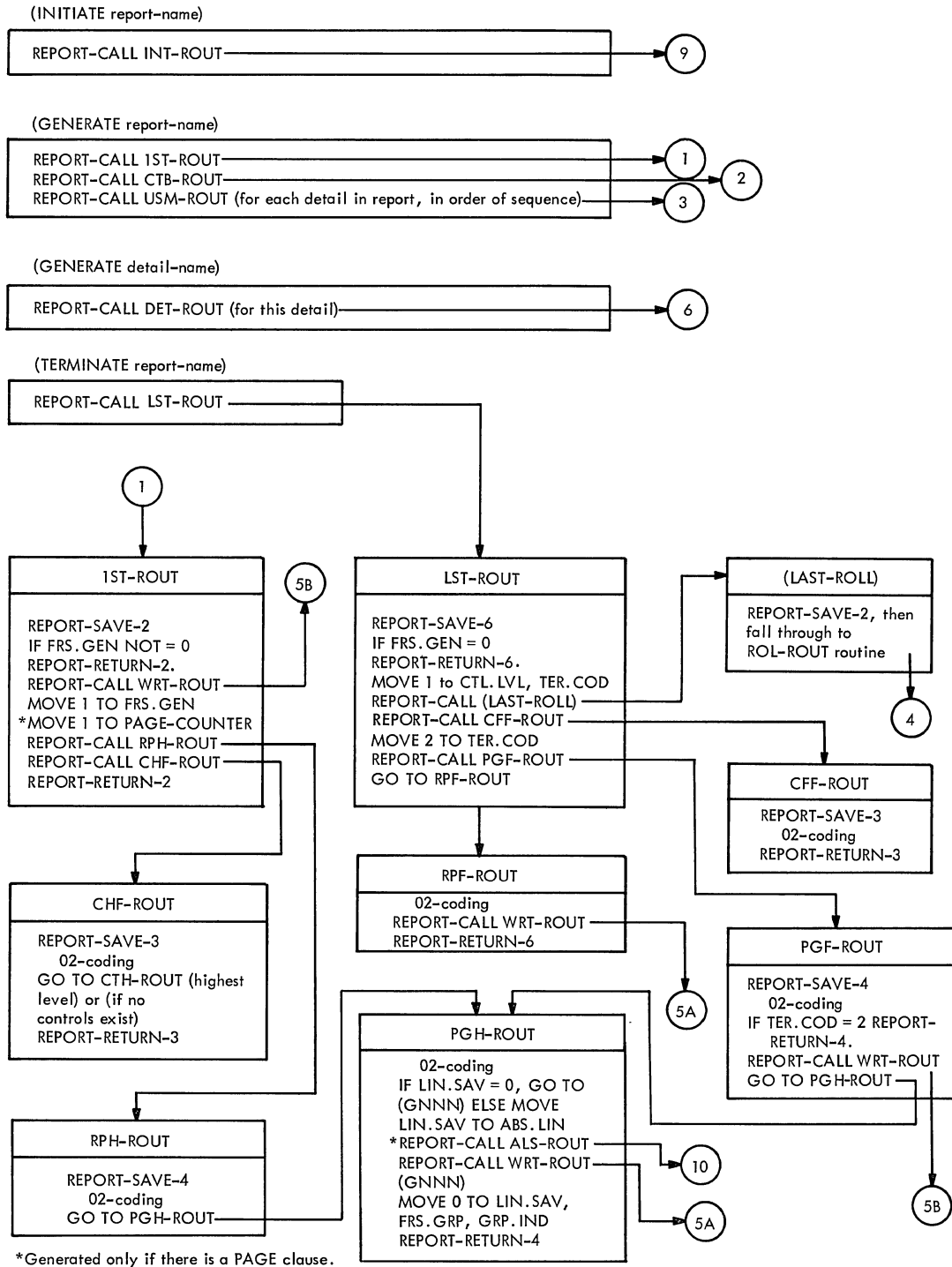
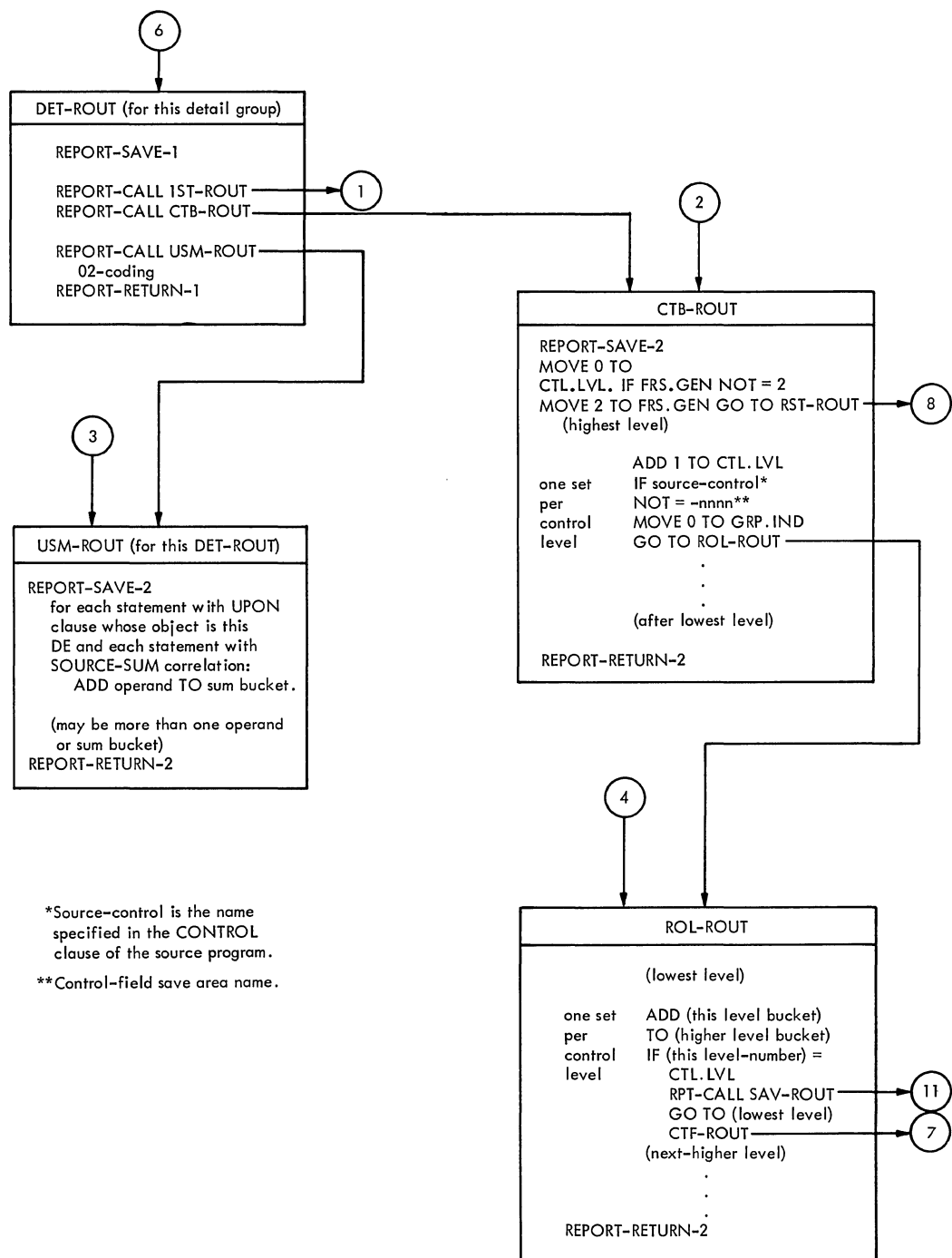


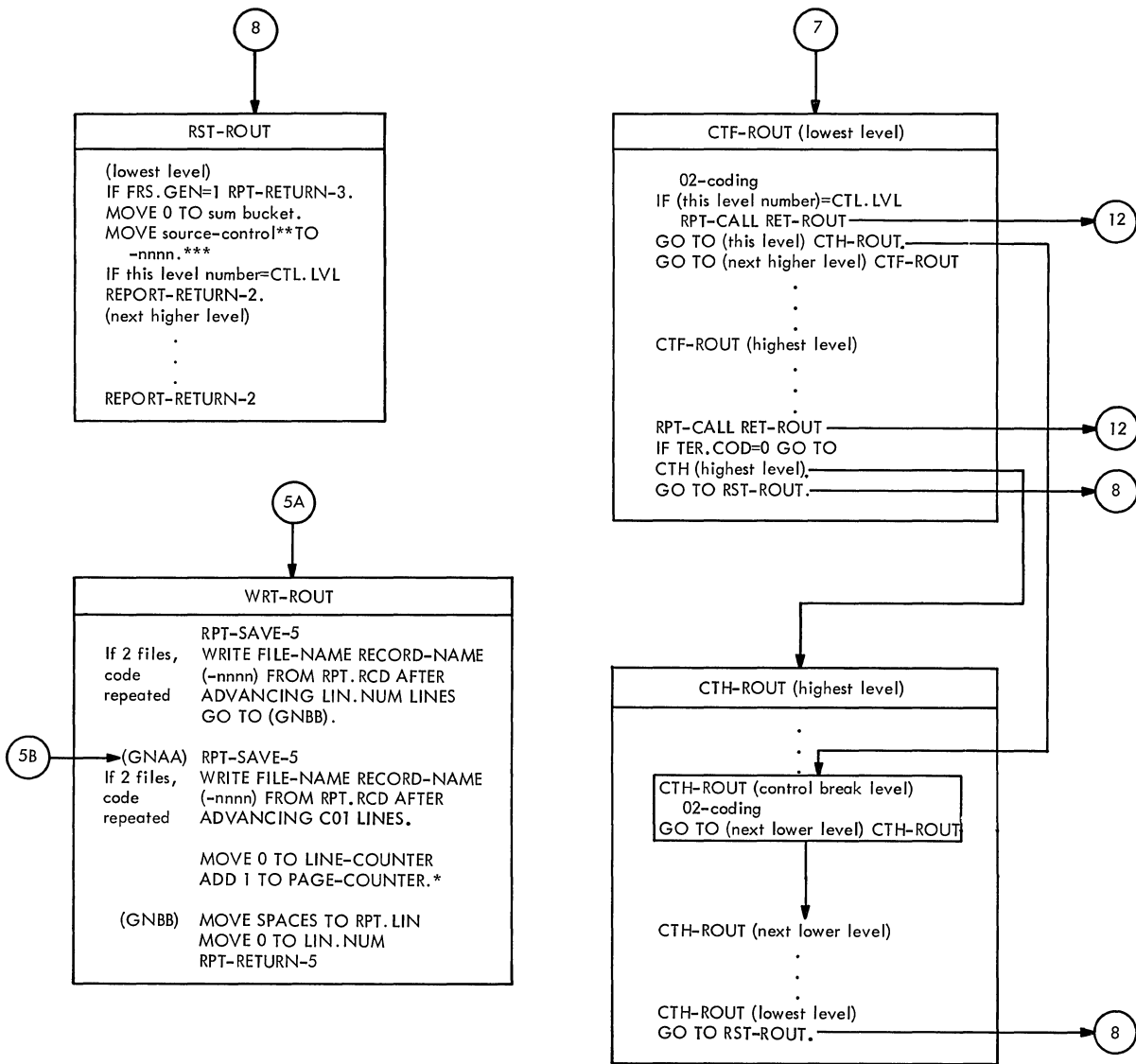
Figure 80. Logic of the Generated Report Writer Subprogram (Part 1 of 4)



\*Source-control is the name specified in the CONTROL clause of the source program.

\*\*Control-field save area name.

Figure 80. Logic of the Generated Report Writer Subprogram (Part 2 of 4)

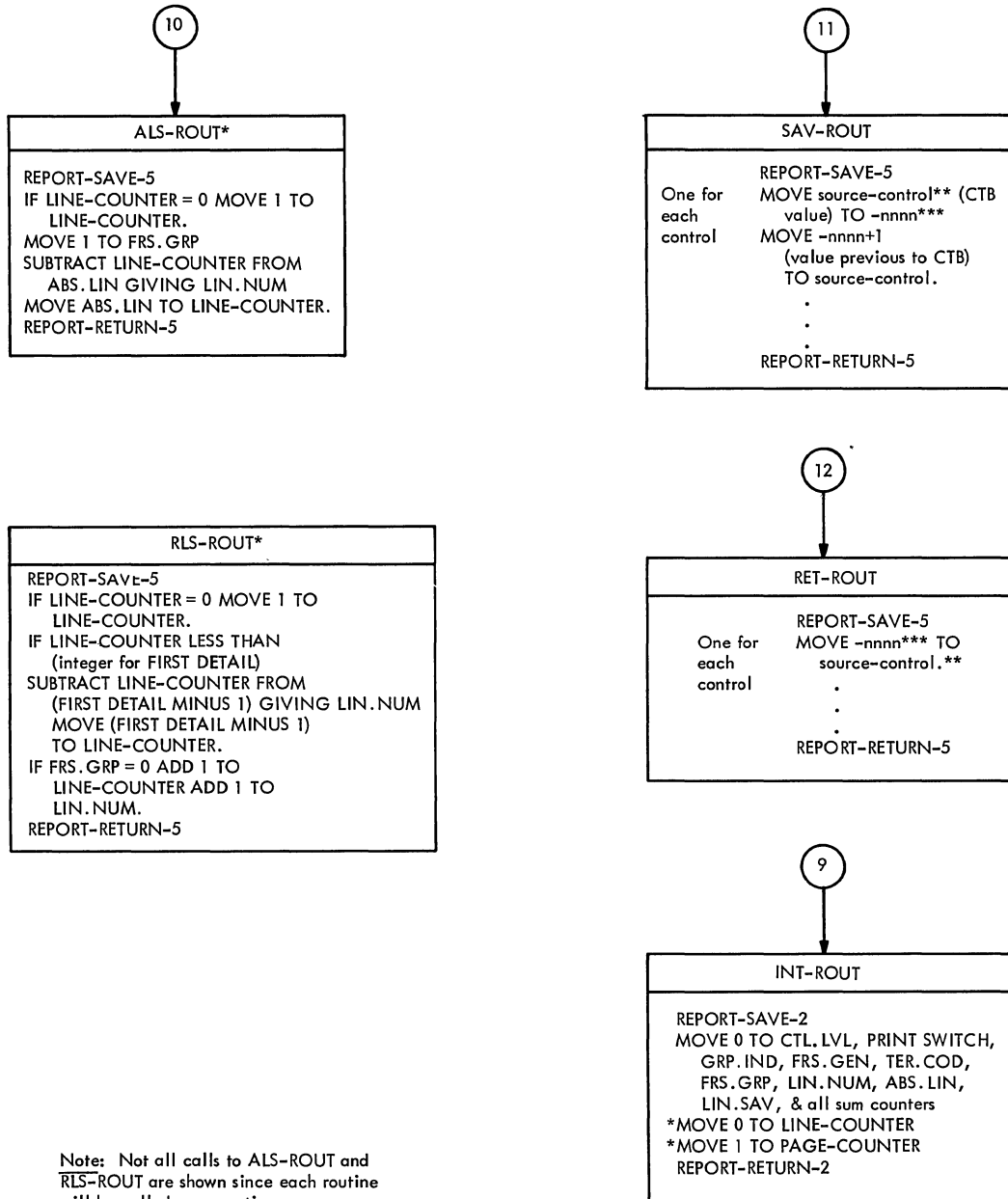


Note: All calls of the WRT-ROUT routine are not shown. Other routines call it as often as necessary to print all output lines to be produced.

- \*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 80. Logic of the Generated Report Writer Subprogram (Part 3 of 4)





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 80. Logic of the Generated Report Writer Subprogram (Part 4 of 4)

### RLS-ROUT Routine

This routine determines the line spacing for relative lines. Routine RLS-ROUT is generated only if the RD entry contains a PAGE LIMIT clause.

### GROUP ROUTINES

Phase 12 generates one group routine for each 01-level record description encountered. The group routine selected is determined by the TYPE clause of the 01-level statement. The coding within the routine varies according to the 01-49-level statements associated with it. An 01-level elementary item contains all necessary information and hence results in a complete group 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 01-level 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.

### RPH-ROUT Routine

This routine produces the report heading. There is one RPH-ROUT routine in an RWS; it results from a TYPE IS REPORT HEADING group.

### RPF-ROUT Routine

This routine produces the report footing. There is one RPF-ROUT routine in an RWS; it results from a TYPE IS REPORT FOOTING group.

### CTH-ROUT Routine

This 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),

there is neither an actual nor a dummy CTH-ROUT routine generated.

### CTF-ROUT Routine

This 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

This 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.

### CFF-ROUT Routine

This 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.

### PGH-ROUT Routine

This routine produces the page headings. There is one PGH-ROUT routine in an RWS; it results from a TYPE IS PAGE HEADING group.

### PGF-ROUT Routine

This 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

This 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-field 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 one-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:

- a. GRP.IND: Group name (01-level) 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.
- b. GP.nnnn: Elementary names (02-level) 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 zero 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 zero, 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 PD that contains a REPORT clause. The size of the 01-level 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. Its PICTURE, however, 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 (4P):** 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 81 and 82 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 83.

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 SYM and 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 field 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	00100	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.

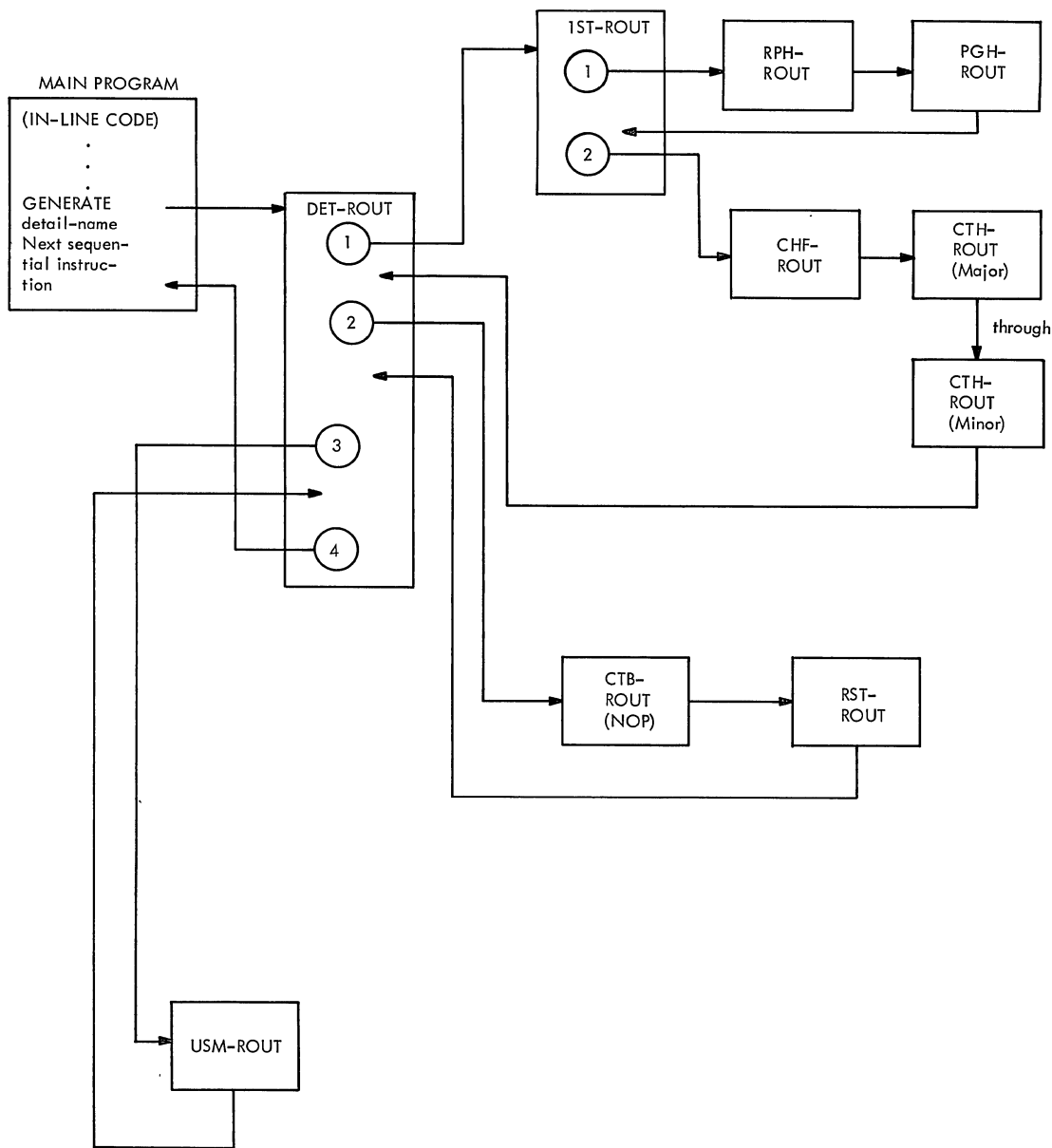


Figure 81. First GENERATE Statement Logic Flow

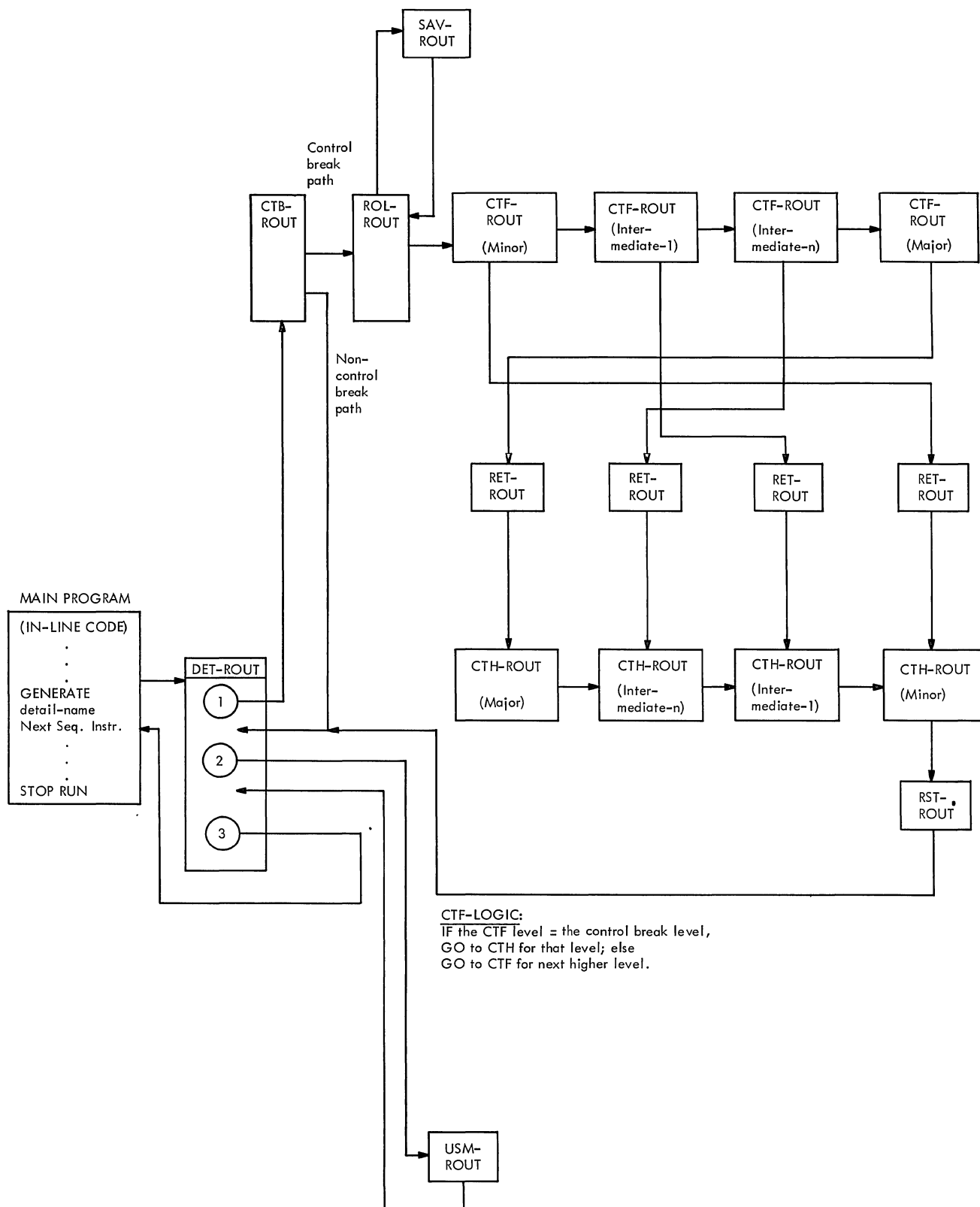


Figure 82. Logic Flow of All GENERATE Statements After the First

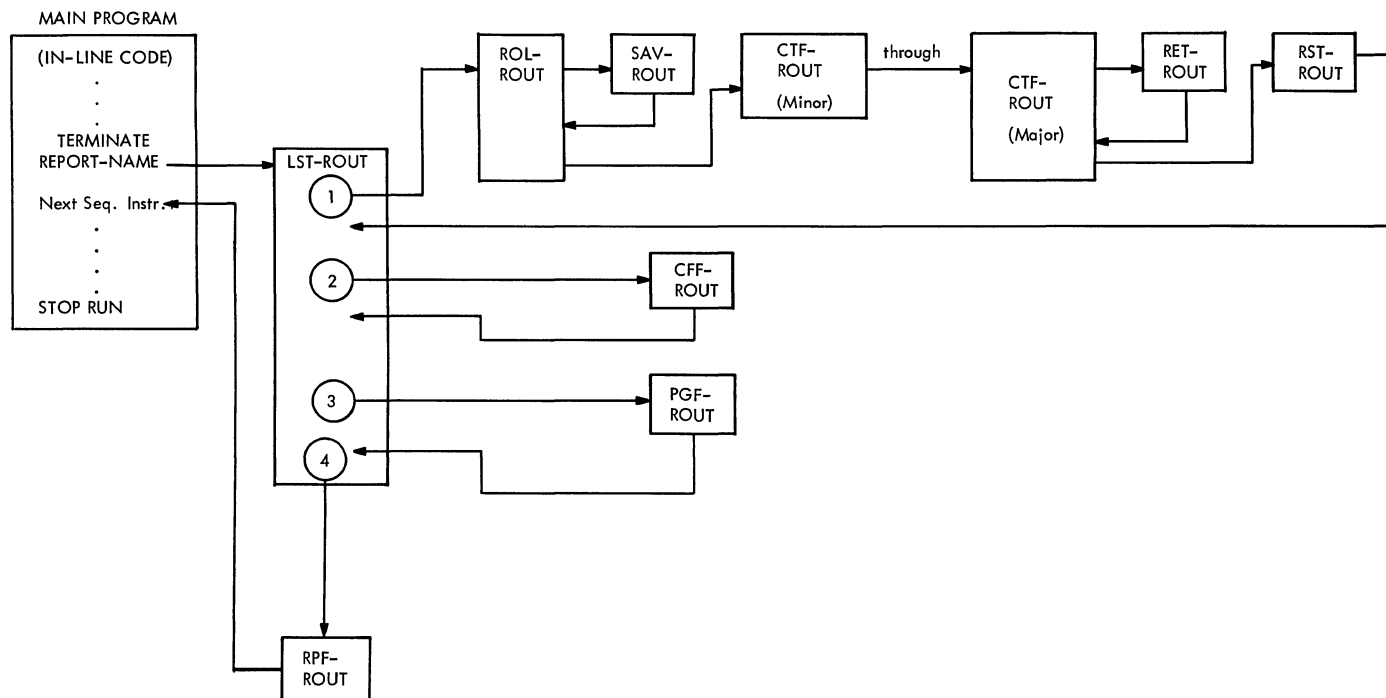


Figure 83. TERMINATE Statement Logic Flow

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 84. 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 84 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 01-level statement containing the TYPE IS DETAIL clause. The generated instruction would look about like this:

GN=032 EQU \*

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	CFF-ROUT
020	INT-ROUT
021	ALS-ROUT
022	RLS-ROUT
023	SAV-ROUT
024	RET-ROUT

Figure 84. RWS GN Numbers



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.

#### 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 80 as a guide.

#### RWS LOGIC FLOWCHARTS

A set of flowcharts for a typical report writer subprogram is included in the Flowcharts chapter of this publication beginning with chart number 501. The logic of many of the routines varies with different source programs. The particular program used to produce this RWS is the report writer program ACME, which is used as the example in the publication IBM DOS Full American National Standard COBOL, Order No. GC28-6394.

The CTF-ROUT routines generated for DAY and MONTH, the two controls used in ACME, are virtually identical; therefore, only the CTF-ROUT flowchart for DAY is included. CTH-ROUT routines were not required for this source program for either control. Phase 12, therefore, generates dummy CTH-ROUT routines; these dummy routines are not included in the set of flowcharts.

APPENDIX D: GENERATED CODE FOR INPUT/OUTPUT VERBS

This appendix shows the generated coding produced by the compiler for input/output verbs. The presentation closely follows the organization of phase 51, where input/output verb coding is generated (see "Input/Output Verbs" in that chapter for further discussion). For each verb, the set of instructions is built in the sequence shown. For example, decision 1 is made and the code shown under it is accordingly skipped or generated, then decision 2, and so on.

OPEN CODING

1. If the file is opened more than one way in a program:

```
MVC  DTFPTR (4) ,SECPTF      Secondary DTF address to current DTF
                                pointer cell
```

2. If OPEN OUTPUT and DTFSD:

```
L      1,DTFPTR              Beginning of DTF
LA     2,224(1)              Get the address of the save area behind
                                the DTF (+224) in order to save the
                                entire DTF in case of subsequent OPEN
                                OUTPUT statements
MVC    0(224,2) ,0(1)        Move the DTF
SH     1,=H'4'               Point to PRE-DTF switch byte
OI     0(1) ,X'80'           Set PRE-DTF switch to indicate that DTF
                                was saved
```

3. If the file is ever closed with lock:

```
L      1,DTFPTR              DTF pointer for subroutine to test the
                                pre-DTF byte switch which indicates
                                whether the file was closed with lock
L      15,=V(ILBDCLK0)        Subroutine to handle error condition
                                (cannot open the file again if
                                pre-DTF switch contains X'FF')
BALR   14,15
```

4. If the STXIT option is requested and there is a standard error declarative for a unit record device:

```
L      15,=V(ILBDABX0)        Subroutine to issue STXIT AB to estab-
BALR   14,15                  lish linkage to the user procedure
```

5. If DTFDA and not relative track addressing, sequential access, OPEN INPUT:

```
L      1,DTFPTR              The three words preceding the DTF (for
SH     1,=H'12'              ERROR, BOF, and EOF addresses) are
XC     0(12,1) ,0(1)         cleared before the OPEN, since
                                different addresses may be used when
                                the file is opened more than one way
                                and there is only one DTF for a
                                direct-access file
```

6. If DTFDA, relative track addressing, sequential, and OPEN INPUT:

L	1,DTFPTR	
SH	1,=H'31'	
MVI	14(1),X'00'	Initialize current-extent bucket
XC	0(3,1),0(1)	Clear TTT of SEEK address
MVI	3(1),X'01'	Record 1
XC	18(12,1),18(1)	Clear PN area

7. If there are any declaratives pertaining to this OPEN:

L	1,DTFPTR	Beginning of DTF
SH	1,=H'nn'	nn=20 for DTFMT, SD (5 possible declaratives) nn=12 for DTFDA (3 possible declaratives) nn=4 for DTFCD, PR, IS (1 possible declaratives)
MVC	0(4,1),PNBOV	Generated only if MT or SD with a BOV declarative
MVC	4(4,1),PNEOV	Generated only if MT or SD with an EOVS declarative
MVC	0(4,1),PNEOF	Generated only if DA with an EOF declarative
MVC	8(4,1),PNEOF	Generated only if MT or SD with an EOF declarative
MVC	4(4,1),PNBOF	Generated only if DA with a BOF declarative
MVC	12(4,1),PNBOF	Generated only if MT or SD with a BOF declarative
MVC	0(4,1),PNERR	Generated only if CD, PR, IS or DU, with an ERROR declarative
MVC	8(4,1),PNERR	Generated only if DA with an ERROR declarative
MVC	16(4,1),PNERR	Generated only if MT or SD with an ERROR declarative

8. If either DTFMT or DTFSD, and there are user-standard labels:

L	1,DTFPTR	PRE-DTF switch will be turned off to
SH	1,=H'4'	indicate to subroutine ILBDUSLO that
NI	0(1),X'DF'	any labels processed are BOF rather than BOV

9. If DTFDA or DTFSD; note that during user standard label processing, a LBRET3 instruction is issued by subroutine ILBDUSLO if labels are being updated. Otherwise, a LBRET1 (no more label processing needed) or LBRET2 (GO TO more labels) is issued, depending on the declarative:

L	1,DTFPTR	If not already done
SH	1,=H'4'	If not already done
OI	0(1),X'40'	Generated only for OPEN I-O
NI	0(1),X'BF'	Generated if not OPEN I-O
OI	0(1),X'08'	Spanned records indicator (generated only for DTFDA)

10. If variable blocked records:

L	1,DTFPTR	Point to PRE-DTF switch if R1 does not
SH	1,=H'4'	already contain this address
NI	0(1),X'EF'	Indicates next WRITE will be the first one

Note: Subroutine ILBDVBLO will not check the space available in the buffer the first time. It will set the PRE-DTF switch on so that, for each subsequent WRITE, the space available will be checked.

11. If DTFDA:

L	4,DTFPTR	
OI	21(4),X'80'	Generated only if OPEN OUTPUT
NI	21(4),X'7F'	Generated only if OPEN INPUT
OI	16(4),X'80'	Generated only if EOP labels

12. If DTFMT and nonstandard labels:

BALR	15,0	Establish addressability
CLI	93(13),X'00'	User-specified number of reels in TGT
BE	28(15)	No - Skip next four instructions
L	1,DTFPTR	Set address of DTF-24, where 2 bytes
SH	1,=H'24'	are set to the reel number for
MVC	0(1,1),93(13)	subroutine ILBDNSL0 to count reels.
MVC	1(1,1),0(1)	

13. If DTFMT, first or only file on reel, and OPEN NO REWIND:

L	1,DTFPTR	Set byte 32, bit 3, in DTFMT to
OI	32(1),X'10'	indicate OPEN NO REWIND

14. If DTFMT and not the first file on the reel:

L	1,DTFPTR	Point to DTF
LA	2, Number of file	Number (sequence) of this file on the reel
L	15,=V(ILBDMFT0)	This subroutine positions the reel
BALR	14,15	to the requested file
OI	32(1),X'10'	Indicate NO REWIND

15. If DTFMT, DTFSD, DTFCD, or DTFPR, opened OUTPUT, and record format F or V:

L	2,BL cell
---	-----------

16. Basic OPEN coding (generated in all cases):

LA	1,=CL8'\$\$BOPEN'	Transient subroutine
L	0,DTFPTR	Point to DTF
LR	4,0	Save DTF address for subroutine
		ILBDUSL0, ILBDNSL0, or ILBDSAEO
CNOP	2,4	Force fullword boundary after the next instruction
BALR	15,0	Establish addressability
ST	0,8(15)	Store DTF address for transient subroutine \$\$BOPEN
BALR	0,12(15)	Branch around DS instruction
DS	1F	To contain DTF address
SVC	2	Fetch transient subroutine

17. If DTFMT:

LA	0,DTFPTR	Point to DTF
L	15,=V(ILBDIML0)	Subroutine to save the PUB number of
BALR	14,15	the first reel in a multiple-reel file

18. If DTFMT and not OPEN NO REWIND:

L	1,DTFPTR	Point to DTF
OI	32(1),X'10'	Indicate NO REWIND for CLOSE time unless REWIND was requested

19. If OPEN OUTPUT, and not single-buffered and not unblocked, and not DTFDA, and if variable records:

- LA 2,4(2) Point to data area of record, by-passing the first 4 bytes (the RECLENGTH field).
20. If OPEN OUTPUT, and not single-buffered and not unblocked, and not DTFDA, and if the records are either fixed, undefined, or variable records with two IOAREAS:
- ST 2,IOPTR Cell If SAME RECORD AREA  
ST 2,BL cell If not SAME RECORD AREA
21. If DTFDA and OPEN OUTPUT:
- L 1,DTFPTR Point to DTF  
L 15,=V(Subroutine) Subroutine to initialize the file by writing R0 onto each track of the specified extents and an EOF record at the end of the last extent, as follows:  
BALR 14,15
- ILBDFMT0, if not relative track addressing  
ILBDRPM0, if relative track addressing.
22. If DTFIS and OPEN OUTPUT:
- L 0,DTFPTR Point to DTF  
LA 1,=CL8'\$\$BSETFL' Transient subroutine to preformat tracks for loading or extending a file  
SVC 2 Fetch transient subroutine
23. If DTFIS and OPEN INPUT and sequential:
- L 1,DTFPTR  
L 15,GN next sentence Sentence beyond fetch, transient \$\$BSETL  
TM 16(1),X'20' Was file assigned IGN  
BCR 1,15 Yes, branch around transient \$\$BSETL, otherwise fall through  
L 0,DTFPTR  
LA 1,=CL8'\$\$BSETL' Transient which initiates the mode for sequential retrieval  
CNOP 2,4 Force fullword boundary after the next instruction  
BALR 15,0 Establish addressability  
ST 0,8(15) Store DTF address  
BAL 0,16(15) Branch around DC instruction  
DC XL4'00' To contain DTF address  
DC CL4'BOF ' Retrieval begins at BOF  
SVC 2 Fetch, transient \$\$BSETL

CLOSE CODING

1. If DTFMT and either CLOSE WITH REWIND or CLOSE WITH LOCK:
- L 1,DTFPTR  
NI 32(1),X'EF' Generated only if CLOSE WITH REWIND  
XI 32(1),X'30' Generated only if CLOSE WITH LOCK  
LR 0,1 Load DTF address into R0

[Go to Number 4]

2. If DTFIS:

[Go to Number 9]

3. If not DTFMT CLOSE WITH REWIND or LOCK:

L 0,DTFPTR

4. Basic coding for all files (CLOSE macro expansion):

```

LR 4,0          Save DTF address in R4 for label
                  subroutines
LA 1,=CL'$$BCLOSE'  Transient subroutine to close file
CNOP 2,4        Force fullword boundary after the next
                  instruction
BALR 15,0       Establish addressability
ST 0,8(15)      Store DTF address
BAL 0,12(15)    Branch around next instruction
DS F            To contain DTF address
SVC 2           Fetch transient subroutine
    
```

5. If DTFMT:

```

L 0,DTFPTR      Transient to shift JIB pointers until
LA 1,=CL8'$$BFCMUL' the PUB pointer of the first volume
                  of the file is found (for
                  multiple-reel files)
SVC 2           Fetch transient subroutine
    
```

6. If DTFMT and CLOSE NO REWIND:

```

L 1,DTFPTR
NI 32(1),X'CF'  Indicate NO REWIND
    
```

7. If DTFSD:

```

L 1,DTFPTR      Point to Pre-DTF switch to determine
SH 1,=H'4'      whether this DTF was saved
BALR 15,0       Establish addressability
TM 0(1),X'80'   Was DTF saved (was it OPEN OUTPUT)
BZ 22(15)       No - Bypass next three instructions
LA 1,4(1)       Point to closed DTF
LA 2,224(2)     Point to save area
MVC 0(224,1),0(2) Move initialized DTF into position for
                  next OPEN OUTPUT
    
```

8. If CLOSE WITH LOCK and file is opened only one way:

```

L 1,DTFPTR
SH 1,=H'4'      Point to Pre-DTF switch
MVI 0(1),X'FF'  Indicate that file must not be reopened
    
```

[Exit - End of CLOSE Coding]

9. If CLOSE WITH LOCK and file is opened more than one way:

```

L 1,SDTFPTR1    Point to first secondary DTF pointer
SH 1,=H'4'      Point to Pre-DTF switch
MVI 3(1),X'FF'  Indicate that file must not be reopened
L 1,SDTFPTR2    Point to second secondary DTF pointer
SH 1,=H'4'      Point to Pre-DTF switch
MVI 0(1),X'FF'  Indicate that file must not be reopened
L 1,SDTFPTR3    Point to third secondary DTF pointer.
                  (Note: skip this and the next two
                  instructions if there is no third
                  SDTF)
SH 1,=H'4'      Point to Pre-DTF switch
MVI 0(1),X'FF'  Indicate that file must not be reopened
    
```

[Exit - End of CLOSE coding]

10. If DTFIS and random access:

[Go to number 3]

11. If DTFIS and never opened as output:

[Go to Number 14]

12. If DTFIS and OUTPUT and two IOAREAs:

L	15,16(1)	Address of logic module
BAL	14,6(15)	ENDFL routine
DC	X'AA00"	

[Go to 13]

13. If DTFIS and OUTPUT and one or two IOAREAs:

L	0,DTFPTR	
LA	1,=CL8"\$\$BENDFL"	Transient subroutine to write last necessary block of data and on EOF record after it
SVC	2	Fetch transient subroutine
LR	1,0	
TM	30(1),X'01'	Check if EOF written
L	15,=V(ILBDISE1)	
BCR	7,15	If EOF record not written, go to ISAM error subroutine to handle

[Go to Number 3]

14. If DTFIS and opened either as sequential input or I-O:

L	1,DTFPTR	
L	15,16(1)	Address of LIOCS module
BAL	14,20(15)	Enter LIOCS module for ESETL (End Sequential Mode)

[Go to Number 8]

15. If RERUN specified for this file:

LA	3,CHKPTCTR	If this is first record, branch around call to
CLI	CHKPTCTR,X'FF'	Checkpoint Subroutine.
L	14,GN=XY	
BCR	8,14	
L	0,CHKPTCTR	Number of records remaining.
L	15,=V(ILBDCKP1)	Checkpoint Subroutine
L	14,GN=XX	If number of records left is not 0, branch around
BCTR	0,14	checkpoint.
BALR	14,15	
DC	AL1(SYSNO)	Parameter for subroutine
DC	CL7'File name'	Parameter for subroutine
GN=XY		
L	0, Number of records	Maximum number of records
GN=XX		
ST	0,CHKPTCTR	Reset counter to maximum number of records.

CLOSE REEL CODING (FOR DTFSD, DTFMT, AND DTFDA SEQUENTIAL INPUT ONLY)

1. If DTFSD and CLOSE UNIT:

CNOP	2,4	Force fullword boundary
L	0,DTFPTR	Address of DTF
LA	1,=CL8"\$\$EOSDEV"	Transient subroutine to handle FEOVD

BALR	15,0	Establish addressability
ST	0,8(15)	Save DTF address in fullword constant
BAL	0,12(15)	Bypass constant, addressing it with R0
DC	F'0'	DTF address
SVC	2	Fetch subroutine \$\$BOSDEV

2. If DTFDA, sequential input:

L	1,DTFPTR	
L	15,=V(Subroutine)	Subroutine to point to beginning of
BALR	14,15	first extent specified on next
		volume, as follows:
		ILBDCRD0, if not relative track
		addressing.
		ILBDRCR0, if relative track
		addressing.

3. If DTFDA, sequential input, with relative track addressing:

L	1,DTFPTR	Address of DTF
L	15,=V(ILBDRCR0)	Subroutine to point to beginning of
BALR	14,15	first extent specified on next
		volume, for relative track addressing

4. If DTFMT:

L	1,DTFPTR	Point to beginning of DTF
LR	4,1	Save DTF address for label subroutine
NF	31(1),X'FD'	Turn off FORCE-REWIND indicator

5. If DTFMT and CLOSE REEL with REWIND:

NI	32(1),X'EF'	Turn off NO-REWIND indicator
----	-------------	------------------------------

6. If DTFMT:

L	15,16(1)	Address logic module
BAL	14,16(15)	Enter module for EOVS processing

7. If DTFMT and CLOSE REEL WITH REWIND:

OI	32(1),X'10'	Turn on NO-REWIND indicator
----	-------------	-----------------------------

8. If DTFMT or DTFSD, and variable records:

LA	2,4(2)	Point to data section of record
----	--------	---------------------------------

9. If DTFMT or DTFSD, and fixed, variable, or undefined records and not single buffered, unblocked records:

ST	2,BL	Save pointer to data record
----	------	-----------------------------

READ CODING

1. If CHKPT was requested for this file:

LA	3,CHKPTCTR	If this is first record, branch
CLI	CHKPTCTR,X'FF'	around call to Checkpoint subroutine
L	14,GN=XY	
BCR	8,GN=XY	
L	0,CHKPTCTR	Number of records remaining
L	15,=V(ILBCKP1)	Subroutine to issue checkpoint macro
L	14,GN=XX	
BCTR	0,14	If count is greater than zero, subtract
		one and skip next four instructions



BALR	14,15	If count equals zero, branch to subroutine to issue CHKPT
DC	AL1(SYSNO)	Parameter for subroutine
DC	CL7'File name'	Parameter for subroutine
GN=XY		
L	0,Number of Records	Maximum number of records
GN=XX		
ST	0,CHKPTCTR	Reset counter

2. Basic coding (generated in all cases):

L	1,DTFPTR	Pointer to beginning of DTF
L	15,GN(EOF)	Address for EOF
TM	16(1),X'20'	Was file assigned IGN
BCR	1,15	Yes - Go to EOF

3. If sequential and not unit record:

LR	4,1	Save DTF address
----	-----	------------------

4. If spanned records and either DTFSD or DTFMT, multiple-reel file labels omitted:

L	0,BL	Address of record
SH	0,=H'4'	Point to RECSIZE field
CNOP	2,4	Force fullword boundary after the BALR instruction
L	5,EOFADDR	User EOF address
BALR	15,0	Establish addressability
ST	5,20(15)	Save EOFADDR in first fullword
ST	0,24(15)	Save WORKAREA adds in second fullword
LA	5,20(15)	Point to first DS
L	15,16(1)	Address of logic module
B	8(5)	Branch around next two constants
DS	F	To contain EOFADDR
DS	F	To contain WORKAREA address
BAL	14,8(15)	Enter logic module

[Go to Number 10]

5. If DTFMT, multiple reel file, labels omitted, and not spanned records:

L	5,EOFADDR	EOF address
BALR	15,0	Establish addressability
LA	3,12(15)	Address of instruction after BAL
L	15,16(1)	Address of logic module
BAL	14,8(15)	Enter logic module

[Go to Number 10]

6. If sequential access:

LA	15,GN(EOF)	At end address
MVC	Disp(3,1),1(15)	Move EOFADDR into DTF
L	15,16(1)	Address of logic module
BAL	14,8(15)	Enter logic module

7. If DTFDA and either random or sequential access:

LA	0,ACTUAL KEY	Generated only if ACTUAL KEY is specified
SR	0,0	Generated only if sequential with no ACTUAL KEY specified

8. If DTFIS and sequential access:

L 0,BL Point to user's record

9. If DTFDA or DTFIS:

L 5,GN(INVALID KEY) Address of INVALID KEY routine  
L 15,=V(Subroutine) Address of subroutine, as follows:  
ILBDSR0, if sequential access, DA,  
and not relative track addressing.  
ILBDRDS0, if sequential access, DA,  
and relative track addressing.  
ILBDDIO1, if random access, American  
National Standard, and not relative  
track addressing.  
ILBDRDI1, if random access, American  
National Standard, and relative  
track addressing.  
ILBDDIO2, if random access, IBM  
American National Standard, and not  
relative track addressing.  
ILBDRDI2, if random access, IBM  
American National Standard, and  
relative track addressing.  
ILBDISM3, if sequential access and  
ISAM.  
ILBDISM2, if random access and ISAM.

BALR 14,15  
[Go to Number 12]

10. If variable records:

LA 2,4(2) Point to data portion of record

11. If two IOAREA's or blocked records, and not SAME RECORD AREA:

ST 2,BL Reset BL to point to buffer in use

12. If DTFIS and SAME RECORD AREA and one IOAREA:

LH 2,100(1) Get address of  
L 2,4(2) IOAREA into R2

13. If either SAME RECORD AREA and not single-buffered, unblocked or ISAM and SAME RECORD AREA:

L 1,FL Point to SAME RECORD AREA  
MVC 0(RECSIZE,1),0(2) Move data from buffer to SAME RECORD  
AREA

14. If file has a subject or an object of an OCCURS clause with a DEPENDING ON option:

L 3,GN=NN NN = GN number for the  
BALR 2,3 Q-Routine for this file

15. Basic coding, generated in all cases:

L 15,GN=NN NN = GN number of next sentence  
BCR 15,15 after AT END or INVALID KEY coding

Note: This is followed by the expansion of the imperative statements following AT END or INVALID KEY.

WRITE AND REWRITE CODING

1. If checkpoint is requested:

LA	3,CHKPTCTR	If this is first record, branch
CL1	CHKPTCTR,X'FF'	around call to Checkpoint subroutine
L	14,GN=XY	
PCR	8,GN=XY	
L	0,CHKPTCTR	Number of records remaining in count
L	15,=V(ILBDCKP1)	Subroutine to issue CKPT macro instruction
L	14,GN=XX	
BCTR	0,14	If number of records remaining is not 0, subtract 1 and skip the next four instructions
BALR	14,15	Go to subroutine
DC	AL1(SYSNO)	Parameter for subroutine
DC	CL7'File name'	Parameter for subroutine
GN=XY		
L	0,Number of records	Load with maximum number of records
GN=XX		
ST	0,CHKPTCTR	Reset counter to maximum number

2. Basic coding (generated in all cases):

L	1,DTFPTB	Address of beginning of DTF
---	----------	-----------------------------

3. If DTFDA:

[Go to Number 27]

4. If DTFIS:

[Go to Number 28]

5. If DTFPR, and single-buffered, unblocked, and an END-OF-PAGE test is required:

OI	2(1),X'04'	Indicate that device end posting is required
----	------------	--

6. If one of the following controls is specified:

a. WRITE {BEFORE or AFTER} {ADVANCING or POSITIONING} identifier

b. WRITE {BEFORE or AFTER} {ADVANCING or POSITIONING} integer-greater-than-3

c. WRITE AFTER {ADVANCING or POSITIONING} with S/360 CTLCHR logic module:

LA	4,Identifier	Generated if Identifier is used in the WRITE statement Generated only if SRA and not MINCASE and not APPLY WRITE ONLY.
L	2,IOPTR	
L	3,BL	
MVC	D(XX,2),0(3)	
L	2,BL	Generated only if not APPLY WRITE ONLY
LA	0,RECORD	Generated only if APPLY WRITE ONLY
LH	3,Length	Length of record
L	15,=V(ILBDSPA0)	Subroutine to issue the I/O requests to the logic module
BALR	14,15	
DC	XL1'Flag'	See Note 1, below
DC	XL1'YY'	See Note 2, below
DC	XL1'ZZ'	See Note 3, below
DC	XL1'XX'	XX is 01 if WITH CODE is specified and 00 if it is not

Note 1:

<u>Bits</u>	<u>Set To</u>	<u>If</u>
0 & 1	00	Binary data-name
	01	Packed data-name
	10	Zoned data-name
2	0	BEFORE
	1	AFTER
3	0	S/360 control characters
	1	ASA control characters
4 & 5	00	Integer
	01	Identifier
	10	Mnemonic-name
6 & 7	Remainder of Integer divided by 3	

Note 2: YY = mnemonic-name equivalent skip/space code, or quotient of integer divided by 3, or length of data-name

Note 3:

<u>ZZ</u>	<u>If</u>
00	Fixed records
01	Variable unblocked records
02	Variable blocked records and not APPLY WRITE ONLY
04	Undefined records
08	Variable blocked records and APPLY WRITE ONLY

[Go to Number 21]

7. If control was specified, and if not DTFC D, and if no subroutine linkage was required:

```
MVI RECORD,NN      NN = control character for LIOCS
```

8. If DTFC D with Mnemonic-name and a control is specified (see note under 9 below):

```
MVI RECORD,NN      NN = control character for LIOCS
```

9. If DTFC D with Identifier and a control character is specified:

```
MVI RECORD,NN      NN = POCKET SELECT 1 or 2
L      2,BL          Pointer to record
MVC   0(1,2),Data-name  Move control character to first byte of
                          record for POCKET SELECT 1 or 2
```

Note: Subroutine ILBDSPA0 is not required except in cases specified under 6 above. The appropriate spacing is placed in the first byte of the record by the code shown, and normal PUT linkage to LIOCS is generated.

10. If spanned records:

```
L      0,BL          Record address
SH     0,=H'4'      Point to record length field in front
                    of record
```

11. If DTFS D:

```
LA     15,GN(INVKEY)  INVALID KEY address
LR     4,1            DTF address
SH     4,=H'4'      Fullword preceding DTF
BALR  14,0           Establish addressability
TM     0(4),X'40'    Was there an OPEN I-O
BO     18(14)        Yes - Skip next two instructions
MVC   NN(3,1),1(15) Move INVALID KEY address into DTF
```

NN = 173 if variable or spanned records  
 NN = 165 if undefined records  
 NN = 161 if Fixed records

OI 16(1),X'01' Indicate end of extent function requested

Note: At end of extent, go to user INVALID KEY address.

12. If SAME RECORD AREA and two IOAREAS and not APPLY WRITE only

a. If record length less than 2047 bytes:

L 2,IOPTR Address of buffer  
 L 3,BL Address of SAME RECORD AREA  
 MVC 0(Length,2),0(3) Move record to buffer  
 MVC 256(length,2),256(3) If greater than 256

b. If record length equal to or greater than 2047 bytes:

L 5,IOPTR Address of buffer  
 L 2,BL Address of SAME RECORD AREA  
 LH 3,=H'RECSIZE'  
 L 15,=V(ILBDMOVO) Move record to buffer  
 BALR 14,15

13. If variable-length and blocked records.

L 2,BL Generated only if not APPLY WRITE ONLY and not SAME RECORD AREA or SAME RECORD AREA and single buffering  
 L 2,IOPTR Generated only if SAME RECORD AREA and double buffering and not APPLY WRITE ONLY  
 LA 2,sending field Generated only if APPLY WRITE ONLY  
 LH 3,=H'RECSIZE' Generated only if not ODOs  
 LH 3,VLC Generated only if ODOs  
 LH 5,=H'SENDING FIELD LENGTH' or 'VLC' Generated only if APPLY WRITE ONLY  
 L 15,=V(ILBDVBL1) Subroutine to write variable-length blocked records  
 BALR 14,15

[Go to Number 22]

14. If variable or undefined records, and not SAME RECORD AREA:

L 2,BL Address of record

15. If variable or undefined records, and they contain an OCCURS...DEPENDING ON clause:

LH 3,VLC Variable-length cell, containing length  
 SH 2,=H'4' Point to record length field  
 LA 3,4(3) Add 4 to include record length field  
 STH 3,0(2) Put length in record length field

16. If variable records and no OCCURS...DEPENDING ON clause:

SH 2,=H'4' Point to record length field  
 LH 3,=H'RECSIZE' Record length given by user  
 LA 3,4(3) Include record length field  
 STH 3,0(2) Store length in Record length field for LIOCS

17. If variable records, or if undefined records with an OCCURS...DEPENDING ON clause:

XC 2(2,2),2(2) Set record halfword of record length field to zero

18. If undefined or spanned records, and no OCCURS...DEPENDING ON clause:

LH 3,=H'RECSIZE' Record length for LIOCS

19. If spanned records:

AH 3,=H'4' Include record length field

20. If any file except DA, IS, or PR with SR linkage:

LR 4,1 If sequential but not unit record, save DTF address for label or error processing subroutine  
L 15,16(1) Address of LIOCS module  
BAL 14,12(15) Branch into logic module

21. If variable and double buffered records:

LA 2,4(2) Point to data, past record length field.

22. If variable and double buffered records and SAME RECORD AREA:

ST 2,IOPTR Save cell for IOAREA being used

23. If variable records not SAME RECORD AREA, and two IOAREAS:

ST 2,BL Address of IOAREA

24. If Q-routines are required:

OI 73(13,X'01' Turn on CALCULATE MAXIMUM RECORD LENGTH indicator  
L 3,GN(Q-RTN) Address of Q-routine for this file  
BALR 2,3  
NI 73(13),X'01' Turn off CALCULATE MAXIMUM RECORD LENGTH indicator

25. If END-OF-PAGE test required:

L 15,GN(Next Sentence) For branch if not EOP  
NI 2(1),X'FB' Turn off DEVICE END posting request  
TM NN(1),X'01' Test for unit exception:

NN = 4 if S/360 control characters  
NN = 39 if ASA control characters

BCR 8,15 Not EOP, branch to next sentence; else, fall through to user's EOP coding

26. If INVALID KEY is used:

L 15,GN(Next Sentence) Address for bypass of INVALID KEY coding  
BR 15 Go to user's next sentence

27. If DTFDA:

<p>LA 0,ACTKEY L 5,GN(INVKEY) L 15,=V(Subroutine)</p>	<p>Point to user's ACTUAL KEY for subroutine Address of user's INVALID KEY coding Address of subroutine, as follows: ILBDDIO0, if American National Standard WRITE and not relative track addressing. ILBDRDIO, if American National Standard WRITE and relative track addressing. ILBDDIO3, if REWRITE and not relative track addressing. ILBDRDI3, if REWRITE and relative track addressing. ILBDDIO4, if IBM American National Standard WRITE and not relative track addressing. ILBDRDI4, if IBM American National Standard WRITE and relative track addressing.</p>
<p>BALR 14,15</p>	

28. If DTFIS and SEQUENTIAL REWRITE:

L 0,=A(RECORD)

29. If DTFIS:

<p>L 5,GN(INVKEY) L 15,=V(Subroutine)</p>	<p>Address of user's INVALID KEY coding Address of subroutine, as follows:  ILBDISM0 if WRITE SEQUENTIAL ILBDISM1 if RANDOM WRITE ILBDISM4 if RANDOM REWRITE ILBDISM5 if REWRITE SEQUENTIAL</p>
<p>BALR 14,15</p>	

[Go to Number 26]

SEEK CODING

1. Basic coding, generated in all cases:

<p>L 1,DTFPTR LR 15,1 SH 15,=H'26' LA 3,ACTUAL KEY MVC 0(7,15),0(3) LA 0,7 L 15,42(15) BALR 14,15</p>	<p>Point to beginning of DTF  Point to PRE-DTF area for SEEK address Move first seven bytes (MBBCCHH) of user's ACTUAL KEY to PRE-DTF area SEEK command code Logic module address (DTF+16) Enter logic module</p>
---	---

START CODING

1. Basic coding, generated in all cases:

<p>L 1,DTFPTR L 15,GN next sentence TM 16(1),X'20' BCR 1,15</p>	<p>Point to beginning of DTF Sentence beyond INVALID KEY coding Was file assigned IGN? Yes. Branch around INVALID KEY coding.</p>
---	---

		Otherwise, fall through.
L	15,16 (1)	Address of logic module for ESET
BAL	14,20 (15)	Enter logic module
L	0,DTFPTR	

2. Generated only if KEY EQUAL TO:

LA	3,data-name	Address of data-name containing key value
LH	5,'LENGTH'	Length of identifier
L	15,=V(ILBDSTRO)	Address of subroutine to move value of data-name to NOMINAL KEY and issue \$BSETL MACRO with 'GKEY' option
BALR	14,15	

3. Generated only if no KEY EQUAL TO:

L	15,=V(ILBDSTR1)	Address of subroutine to issue \$BSETL MACRO with 'KEY' option
BALR	14,15	
TM	30(1),X'D8'	Test for SETL errors
L	15,GN Next sentence	Sentence beyond INVALID KEY coding
BCR	8,15	No errors; else, fall through to user's INVALID KEY coding

DISPLAY CODING

The coding generated for a DISPLAY statement consists of linkage to the DISPLAY subroutine or, when the OPT option has been specified, the Optimizer DISPLAY subroutine (ILBDDSS0). The various forms of that linkage are discussed in the description of those subroutines in the publication IBM DOS/VS COBOL, Subroutine Library, Order No. LY28-6424.

ACCEPT CODING:

1. Basic coding, generated in all cases:

L	15,=V(ILBDACP0)	Subroutine to read a record from SYSIPT or console
BALR	1,15	
DC	XL2'Device Code'	The device codes are: X'0002', if CONSOLE X'0004', if SYSIPT
DC	XL1'TYPE'	The bits have the following meanings Bit 0-1: (not used) 2 Variable Length 3 : Pointer ADCON is direct 4-7: (not used)
DC	XL3'MNN'	If binary or internal decimal, M=length of input item and NN=length of converted result If variable length, MNN=the address of the VLC-cell; otherwise, MNN=length of the item
DC	AL4(Base locator)	Or AL4(operand-text) if bit 3 of TYPE is set
DC	XL2'Displacements'	Displacement of text from base



USE CODING

1. If USE...ERROR:
 

ST	14,SA2	Return address
CNOP	2,8	Force doubleword boundary after branch instruction
BALR	15,0	Establish addressability
B	12(15)	Skip eight bytes
  
2. If USE...ERROR and GIVING...Option 1 (error bytes):
 

LA	2,Data-name	User's error byte area
CNOP	0,8	Position to next doubleword boundary
  
3. If USE...ERROR and not GIVING...Option 1:
 

DC	XL8"00"	Indicates no request for error bytes
----	---------	--------------------------------------
  
4. If USE...ERROR and GIVING...Option 2, i.e., data-name-2:
 

ST	1,BLL=N	Point to error block. If D-N is in working storage, n=1; if D-N is in the linkage section, n=the number of the next available BLL.
MVC	Data-name,0(1)	Move to user-defined area. This statement is generated by Phase 40 The length of the buffer is the "TO" field of the operand; of the length of the record is the "FROM" field
  
5. If USE...ERROR:
 

L	14,SA2	Restore return address
BR	14	Return to subroutine ILBDSAE0, ILBDDAE0, or ILBDISE0
  
6. If USE...LABELS:
 

ST	2,BLL=1	SAVE address of label
ST	4,SA2	SAVE DTF address
  
7. If USE...LABELS and nonstandard labels:
 

L	4,SA2	
L	15,=V(ILBDNSL1)	Subroutine which will return control to user's procedure
BR	15	
  
8. If USE...LABELS and user-standard labels:
 

L	4,SA2	
L	15,=V(ILBDUSL1)	Subroutine which will return control to user's procedure
BR	15	
  
9. If GO TO MORE-LABELS and non-standard labels:
 

L	4,SA2	Restore DTF address
L	15,=V(ILBDNSL2)	Subroutine to return control to LIOCS to read or write the next label
BR	15	
  
10. If GO TO MORE-LABELS and user-standard labels:
 

L	4,SA2	Restore DTF address
L	15,=V(ILBDUSL2)	Subroutine to return control to LIOCS to read or write the next label.
BR	15	

## GLOSSARY

The words listed below are defined according to their use in this book, and the definitions are not necessarily applicable elsewhere. Efforts have been made to exclude terms which are common in the programming profession unless they are used in a special sense.

Associated file: A file on a 3525 card punch device with optional read and print feature for which more than one of the read, punch, or print functions has been specified in the ASSIGN clause of the SELECT sentence.

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."

ACCESS routines: A group of routines which build and access the dictionary. They reside in storage as part of Phase 00. See "Appendix A."

Base Locator (BL): A 4-byte address cell in the TGT. There is one BL pointing to the Report Section, one to the Working-Storage Section, and one to each FD, SD, and RD entry. Any FD, SD, or RD entry exceeding 4,096 bytes will have one BL assigned to each 4,096 bytes. Phase 60 loads a register with each address unless there are too many BLs. In that case, it loads registers with BLs as they are needed. BLs are assigned in phase 22.

Base Locator for Linkage Section (BLL): A 4-byte address cell in the TGT. BLLs are assigned by counter and are unique. BLL1 points to a work area used to process label records. BLL2 through BLLn are assigned to each 77-level and each 01-level entry in the Linkage Section. Any 77- or 01-level entries exceeding 4,096 bytes have one BLL assigned per 4,096 bytes. BLLs are assigned in phase 22.

BL: see Base Locator.

BLL: see Base Locator for Linkage Section.

COBOL Library Subroutine: One of a set of subroutines that perform frequently required operations and which, because they are too extensive to be efficiently placed into the object module wherever needed, are stored in the COBOL library and included in the load module by the linkage editor when needed. See the publication IBM DOS/VS COBOL Subroutine Library Program Logic, Order No. LY28-6424.

COBOL space: The difference in length between the longest phase and the phase currently processing. This space is available to supplement the space initially assigned to TAMER.

COMMON: A communications region available to all phases for storing and accessing information. It is resident in storage at the beginning of phase 00 and accessible via DSECTS to other phases. See "Section 5. Data Areas."

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.

Data A-text: An internal compiler text generated by phases 21 and 22. It is used by phase 60 to generate the data and global table areas of the object module. See "Section 5. Data Areas."

Data IC-text: An internal compiler text generated by phases 10 and 12. It is used in phases 21 and 22 to create Data A-text. See "Section 5. Data Areas."

Debug text: A type of debugging text which contains card numbers, their displacement within the object module, the priority of each segment, and discontinuity elements.

DEF-text: An internal compiler text generated in phases 22 and 30. It is used by phase 61 to create the cross-reference table. See "Section 5. Data Areas."

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 dictionary pointer to the next item with the same or a lower level number.

Dictionary: A table, built by phases 11, 21, and 22, in which is stored information about procedure-names and data operands. When SYMDMP is in effect, it is used by phase 25 to build the Debug File. It is used by phase 30 when creating P1-text and then released. The dictionary format is shown in "Section 5. Data Areas."

Dictionary attributes: Descriptive information about a source program-name placed in the dictionary by phases 11, 21, and 22. Phase 30 replaces each name with its dictionary attributes.

Dictionary pointer: The pointer to the location of an entry in the dictionary. Since it provides a unique identification for a name, it is retained and used by some phases after the dictionary has been released.

E-text: An internal compiler text generated by phases 10 through 51 whenever an error is detected. It is used by phase 70 to generate error messages. See "Section 5. Data Areas."

Fixed routine: One of a set of routines generated by phase 12 as part of the Report Writer subprogram. Fixed routines never vary in logic content. See Appendix C.

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 SYMDMP or STATE option.

GN: see Procedure-name.

Group routines: One of a set of routines generated by phase 12 as part of the Report Writer subprogram. There is one group routine for each 01-level statement in the Report Section. See Appendix C.

Incomplete Data A-text: An internal compiler text generated by phase 20. It has a blank location field, later filled in by phase 22, and a 2-byte prefix. Otherwise, its format is that shown for Data A-text in "Section 5. Data Areas."

Initialization routines: Collectively, routines INIT1, INIT2, and INIT3. These are generated by phase 60 as part of the object module. A discussion of them, including their coding, is provided in Appendix B.

Intermediate A-text: An internal compiler text generated by phase 50. It may be basically Procedure A-text or Optimization A-text and has an identification prefix attached for phase 51. See "Section 5. Data Areas."

Intermediate E-text: An internal compiler text generated by phase 50. It consists of E-text to which an identification prefix has been added for phase 51. See "Section 5. Data Areas."

Main Free Area: Space in storage permanently allocated for tables and the dictionary. It is located immediately after (that is, in next higher location of storage after) COBOL space.

Major code: A 4-bit code identifying the different types of dictionary entries. The codes are listed in "Section 5. Data Areas."

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 an OCCURS clause with the DEPENDING ON option. See the "Phase 25" and "Phase 65" chapters and "Section 5. Data Areas."

Minor code: A 4-bit code identifying the type of operand in the dictionary entry of an LD item. The codes are listed in "Section 5. Data Areas."

Object hierarchy: The order of all group and elementary items defined within the second group item in a MOVE CORRESPONDING statement.

Object module: The result of a successful compilation. It is the output of a single execution of the compiler and is the input to the linkage editor.

Optimization A-text: An internal compiler text generated by phase 51. It is used by phase 60 to eliminate storage duplications. See "Section 5. Data Areas."

Parametric routine: One of a set of routines generated by phase 12 as part of the Report Writer subprogram. See Appendix C.

PGT: see Program Global Table.

PN: see Procedure-name.

Priority: see Segmentation.

Procedure-name (GN, PN, or VN): The name of a point in a program which can be the object of a branch instruction. PNs are user-assigned procedure-names which correspond to paragraph or section names in the Procedure Division of the source program. GNs are compiler generated and are inserted wherever a need for an additional name occurs. VNs are variable names; that is, they may vary at execution time because of a PERFORM or ALTER statement. All procedure-names are unique since they include a number assigned by a counter (for example, PN1, VN2, GN1, and GN2). See the chapter on phase 40 for a fuller discussion of procedure-names.

Procedure A-text: An internal compiler text generated by phase 51. It is similar to assembler language and is used by phase

60 to create machine code for the object module. See "Section 5. Data Areas."

Procedure Block: Unit of addressability in the machine language program which is produced when OPT is specified. 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: One of a series of three texts, P0, P1, and P2, generated by phases 11, 12, 21, and 22 (P0-text), 30 (P1-text) and 40 (P2-text). They are based on the source program Procedure Division and Report Section statements, and represent stages in the translation process from source statement to Procedure A-text. See "Section 5. Data Areas."

Program Global Table (PGT): A part of the object module. The PGT contains virtuals, literals, and addresses used during execution. See Appendix B.

P0-text: See Procedure IC-text.

P1-text: See Procedure IC-text.

P2-text: See Procedure IC-text.

Q-routine: One of a set of routines generated by phase 22. Q-routines calculate the length of variable-length fields and the location of variably located fields resulting from an OCCURS clause with the DEPENDING ON option.

REF-text: An internal compiler text produced by phase 60 if the SXREF or XREF option is in effect. Phase 61 uses REF-text and DEF-text to produce a cross-reference listing. Each element of REF-text consists of a user-assigned name and the card number of a statement that included that name.

Root segment: See Segmentation.

SBL: See Secondary Base Locator.

Secondary Base Locator (SBL): A 4-byte address cell in the TGT. Phase 22 assigns a unique SBL, using a counter in COMMON, to each variably located field. At execution time, each SBL points to its field. Variably located fields are those which follow a variable-length field and which are not new files or records; they occur as a result of OCCURS...DEPENDING ON statements in the source program. If a variably located field exceeds 4,096 bytes, phase 22 assigns one SBL to each 4,096 bytes.

Section: A series of source program procedure instructions grouped under the same section-name.

Segment: A section or a group of sections all having the same priority.

Segmentation: A special feature of the compiler which permits the programmer to organize his program into several load modules. Each section in the Procedure Division is assigned a priority number. All sections having the same priority are loaded together as a segment. One of these, the root segment, resides in storage throughout execution of the program. The other segments are loaded in order of the priority number, each segment overlaying the one before.

Subject hierarchy: The order of all group and elementary items defined within the first group item to appear in a MOVE CORRESPONDING statement.

Table: An area in storage containing a number of entries of a fixed, often identical, format. Many tables used within a single phase and all tables used by two or more phases are handled by the TAMER routines of phase 00.

TAMER: A set of routines, resident throughout compilation as part of phase 00 and accessible to all phases, which get space for, access, and build tables. Those tables which employ TAMER routines include those passed between phases and those used by a single phase and having a variable length. Usually, tables which have a fixed length and which are built, used, and released within one phase are not handled by TAMER routines.

TAMER space: Space in storage occupied by TAMER tables. It follows (that is, is in a higher location of storage COBOL space).

Task Global Table (TGT): A part of the object module. The TGT contains information, addresses, and work areas for use during execution. See Appendix B.

TIB (Table Information Block): One of thirty-six 8-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.

TGT: See Task Global Table (TGT).

Transient area: A portion of storage reserved during execution time to contain segments which are not permanently resident. It contains one such segment at a time and is large enough to hold the largest nonresident segment in the program.

UPSI: User Program Status Information. There are eight 1-bit UPSI switches provided by the DOS/VS system. The UPSI feature of this compiler provides the facility of naming and using these switches. Phases 10 and 22 perform the processing of this feature.

Verb string: A verb and its operands.

Virtual: The name of a procedure switch 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 reference to a COBOL library subroutine. At execution time, the address of all procedures referred to by virtuals (which have been link edited into the load module) are stored in the Program Global Table.

VN: See Procedure-name.

DIAGRAMS







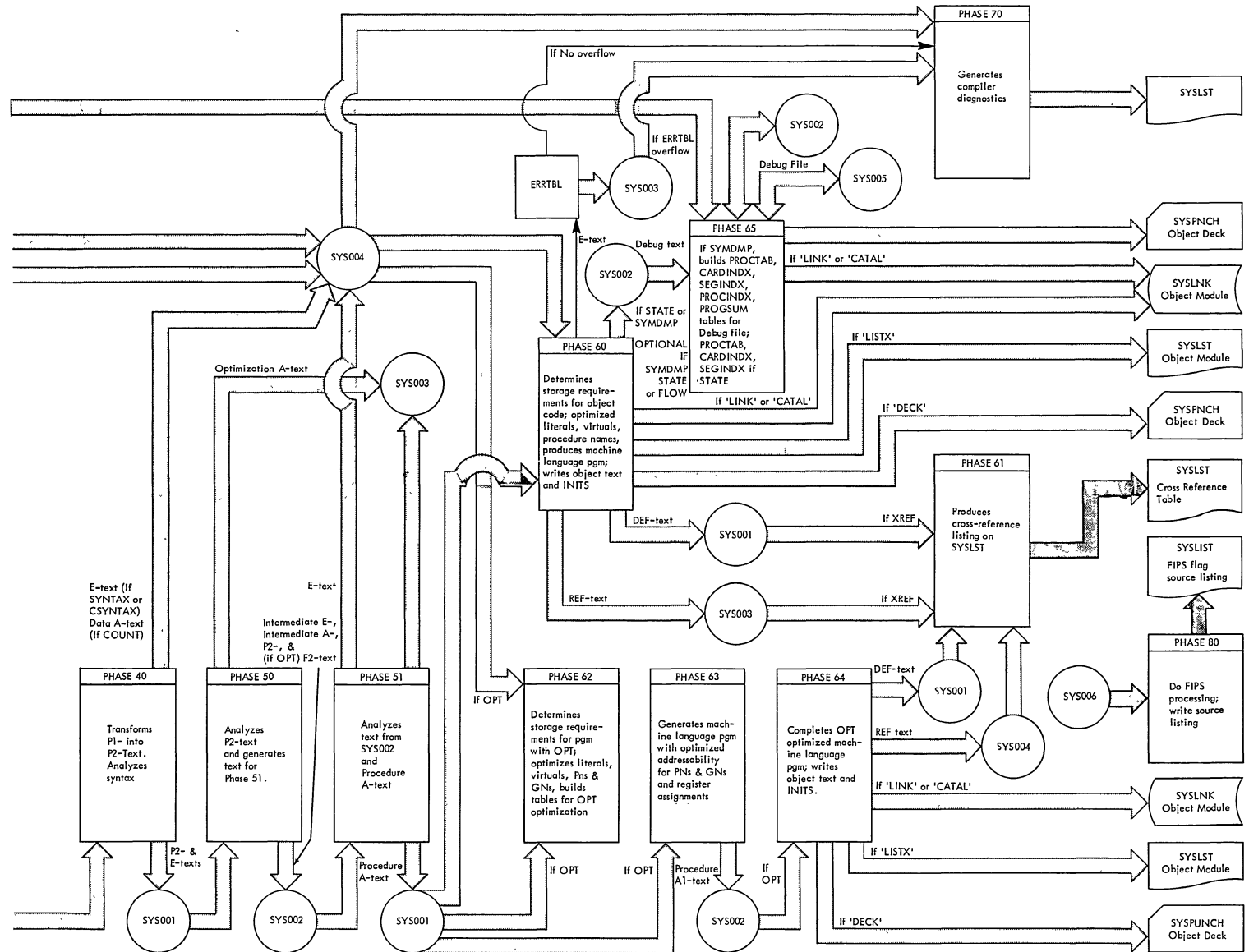


Diagram 1. Overview of the Compiler (Part 2 of 2)



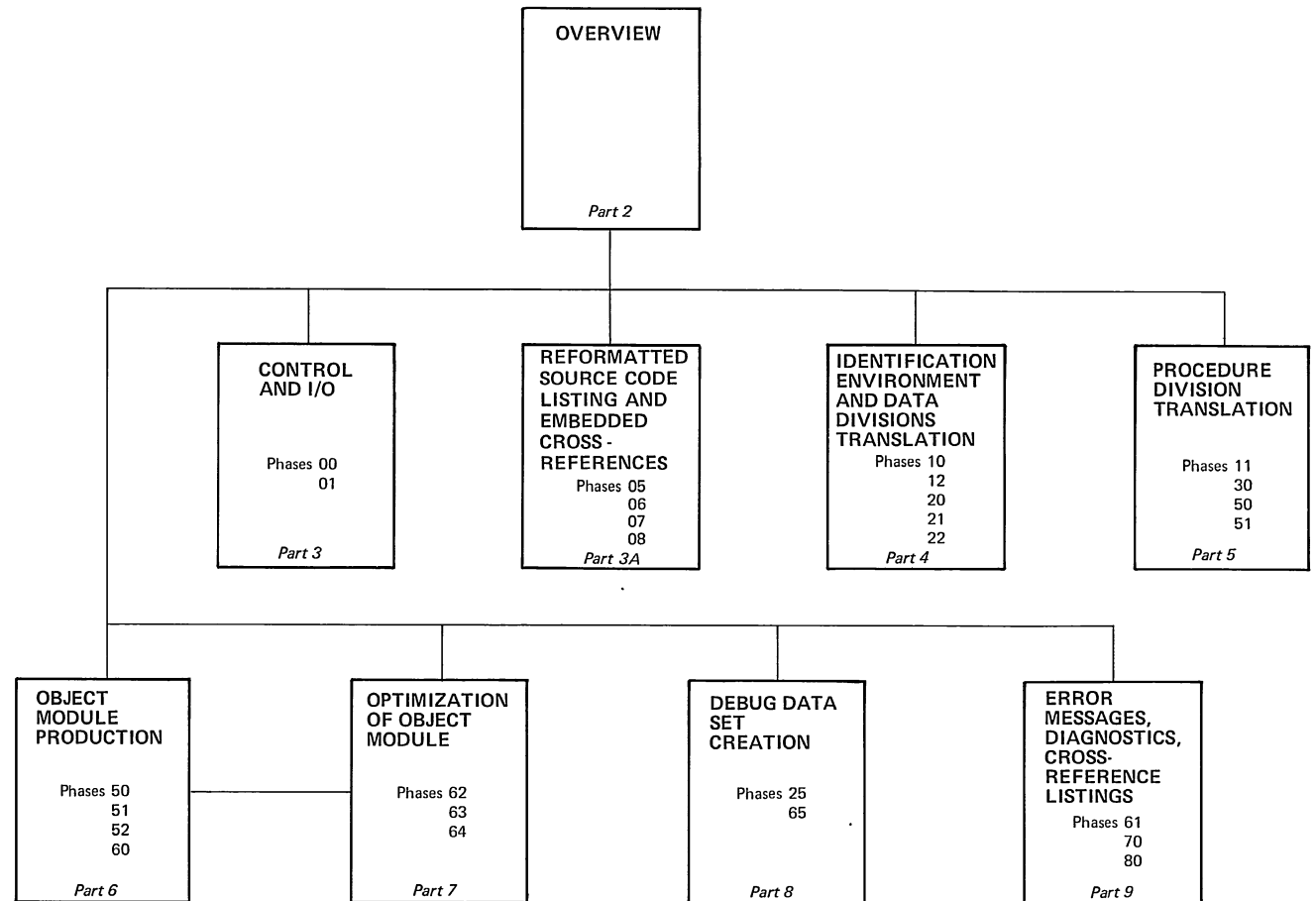


Diagram 2. Part 1. Method of Operation: Table of Contents



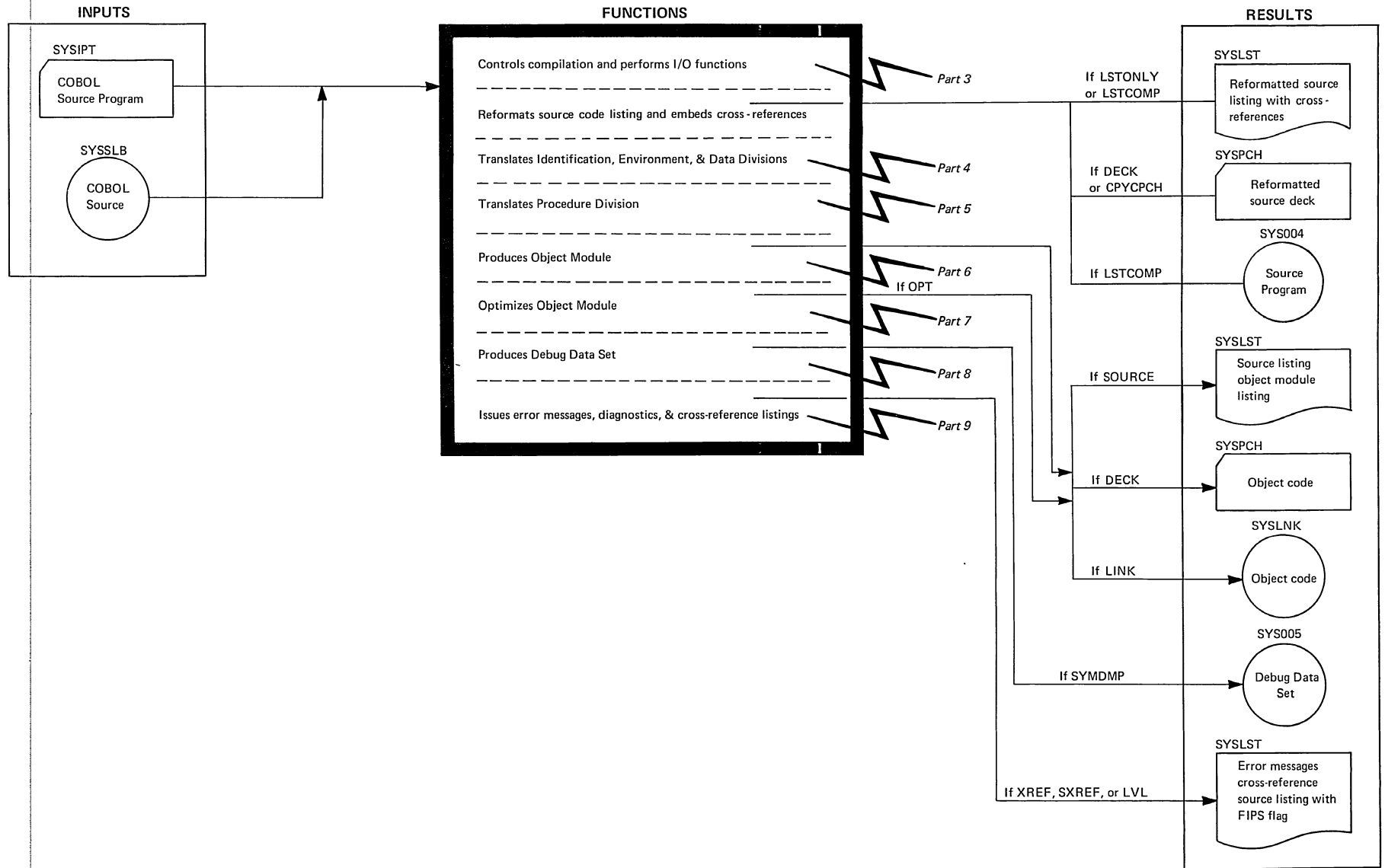
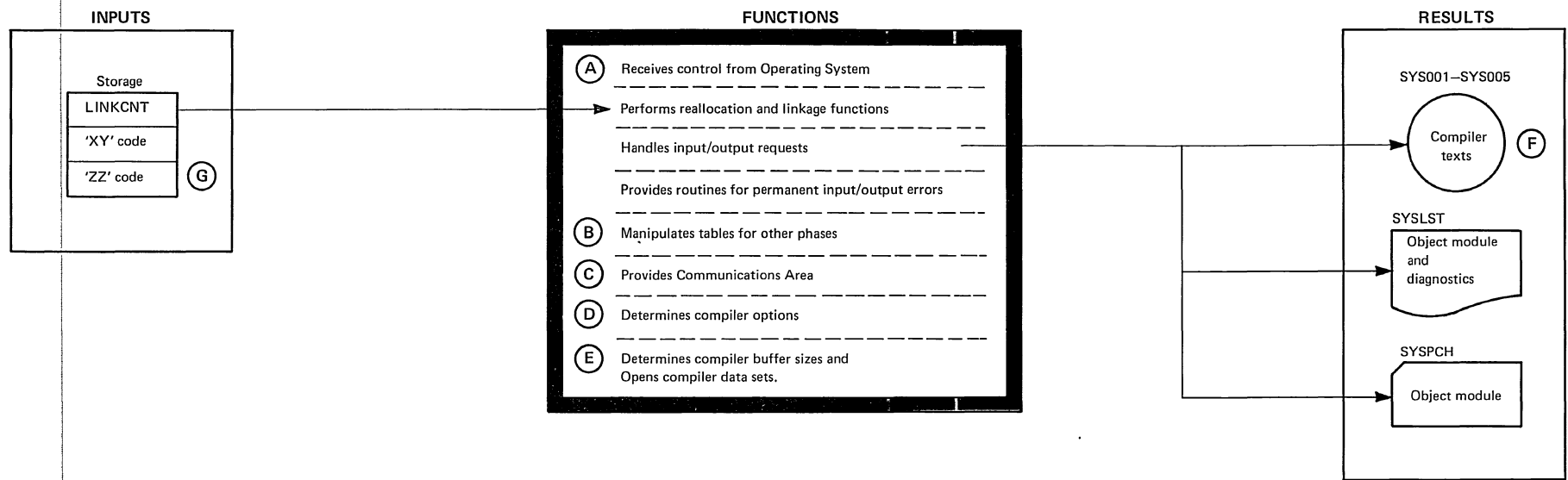


Diagram 2. Part 2. Method of Operation: Overview



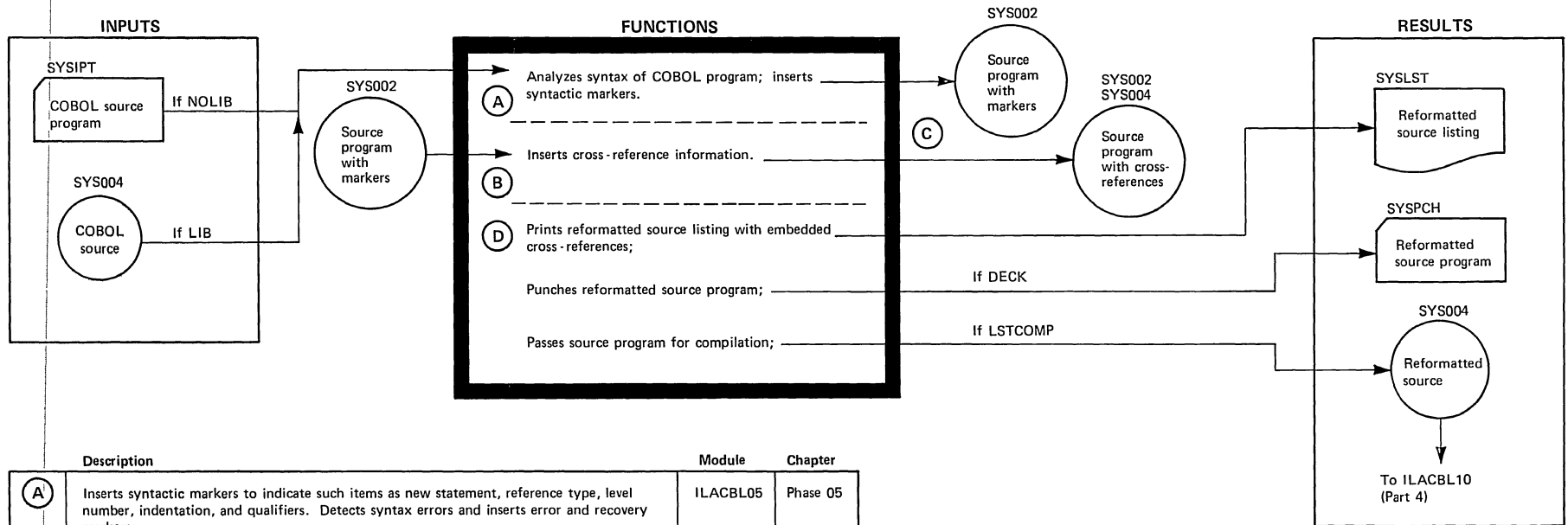


	Description	Module	Chapter	Chart
(A)	Receives control from; returns control to operating system.	ILACBL00	Phase 00	AA
(B)	Provides Table Area Management Executive Routines (TAMER) for acquiring storage for and building tables.	ILACBL00	Table and Dictionary Handling	
(C)	Provides Communications area (COMMON) used to pass information between phases.	ILACBL00	Section 5: Communications Area (COMMON)	
(D)	Contains installation default values of compilation parameters, determines user options. If LIB is specified, handles COPY and BASIC functions.	ILACBL01	Phase 01	BA
(E)	Processes compilation parameters, determines buffer sizes for all phases, obtains storage for tables, dictionary, and buffers, enters information in COMMON, opens data sets.	ILACBL01	Phase 01	BA
(F)	Activity of data sets and buffer assignments is summarized in Figure 10.			
(G)	'XY' and 'ZZ' codes are summarized in Figure 5.			

Diagram 2. Part 3. Method of Operation: Control and Input/Output



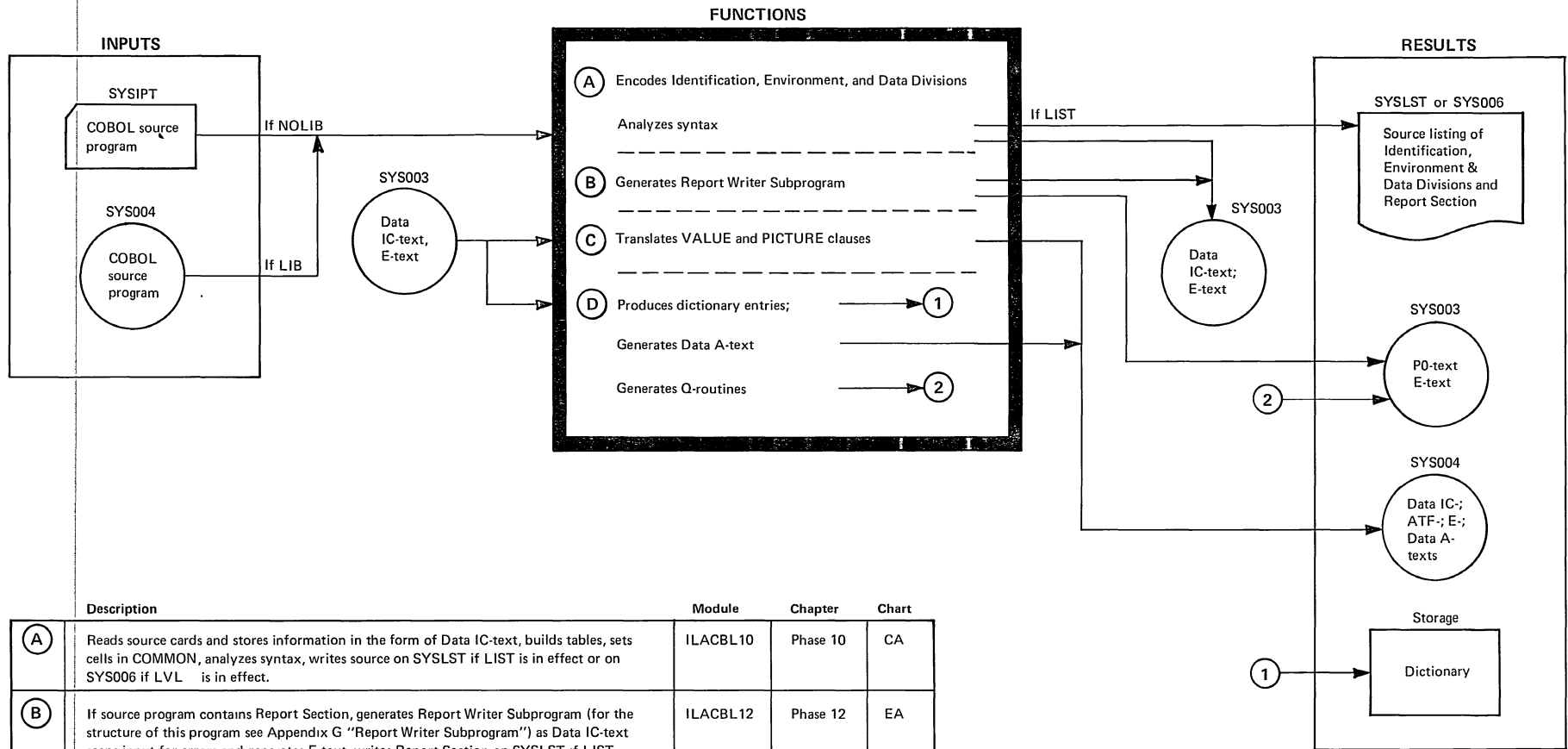




	Description	Module	Chapter
(A)	Inserts syntactic markers to indicate such items as new statement, reference type, level number, indentation, and qualifiers. Detects syntax errors and inserts error and recovery markers.	ILACBL05	Phase 05
(B)	Makes two or more passes of input source; inserts pointers at place of definition to places of reference and at place of reference to place of definition.	ILACBL06	Phase 06
(C)	Output is written alternately on SYS004 and SYS002 at each pass of input source. Output of the last pass is always written on SYS002.		Phase 06
(D)	Prints preface consisting of format description, statement number uses, footnote use, method of indentation, summary listing; lister option listing; punches source program for later compilation if the DECK option is in effect; passes source program to phase 10 if the LSTCOMP options is in effect.	ILACBL07 ILACBL08	Phase 07 Phase 08

Diagram 2. Part 3A. Reformatted Source Code Listing and Embedded Cross-references

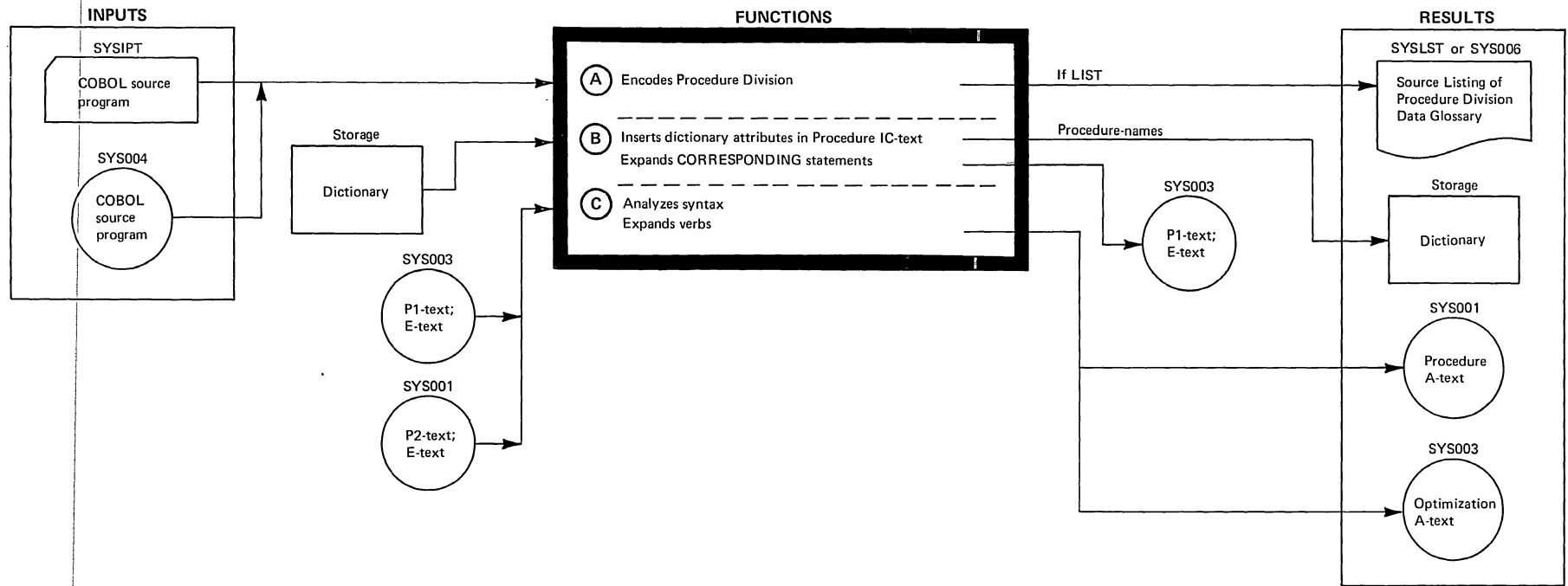




Description	Module	Chapter	Chart
(A) Reads source cards and stores information in the form of Data IC-text, builds tables, sets cells in COMMON, analyzes syntax, writes source on SYSLST if LIST is in effect or on SYS006 if LVL is in effect.	ILACBL10	Phase 10	CA
(B) If source program contains Report Section, generates Report Writer Subprogram (for the structure of this program see Appendix G "Report Writer Subprogram") as Data IC-text scans input for errors and generates E-text, writes Report Section on SYSLST if LIST is in effect (or on SYS006 if LVL is in effect), builds TAMER tables and sets COMMON cells. If VERB is in effect, generates Listing A-text.	ILACBL12	Phase 12	EA
(C) Translates VALUE and PICTURE clauses from Data IC-text to ATF-text, writes partial dictionary entry for each LD, scans input for errors and generates E-text, builds TAMER tables.	ILACBL20	Phase 20	FA
(D) Produces dummy FD and SD dictionary entries, builds CD, LD, RD, and ID entries, completes Data A-text, generates Q-routines, produces E-text, builds TAMER tables; completes FD and SD entries, writes Data A-text for DTFs, determines buffer sizes, produces E-text.	ILACBL22	Phase 22	HA
	ILACBL21	Phase 21	GA

Diagram 2. Part 4. Method of Operation: Identification, Environment, Data Division Translation

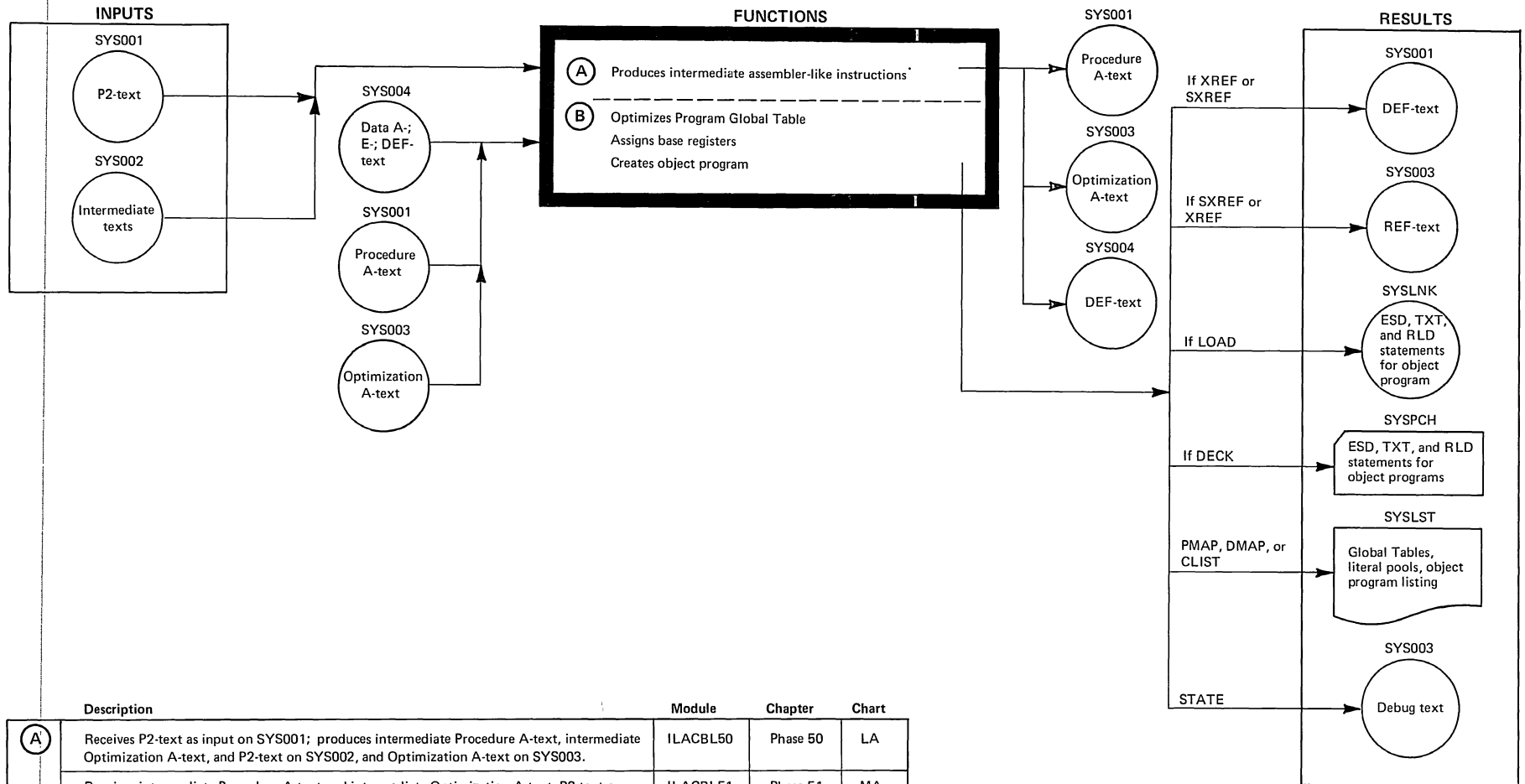




	Description	Module	Chapter	Chart
(A)	Reads Procedure Division of source program; creates dictionary entries for procedure-names; writes Procedure Division on SYSLST if LIST is in effect (or on SYS006 if LVL is in effect); processes Declaratives Section; generates Listing A-text if VERB is in effect.	ILACBL11	Phase 11	DA
(B)	Creates 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.	ILACBL30	Phase 30	JA
(C)	Translates P1-text from SYS003 to P2-text on SYS001.	ILACBL40	Phase 40	KA
	Processes arithmetic verbs from SYS001 to SYS002.	ILACBL50	Phase 50	LA
	Processes input/output verbs and other non-arithmetic verbs from SYS002 to SYS001.	ILACBL51	Phase 51	MA

Diagram 2. Part 5. Method of Operation: Procedure Division Translation





	Description	Module	Chapter	Chart
A	Receives P2-text as input on SYS001; produces intermediate Procedure A-text, intermediate Optimization A-text, and P2-text on SYS002, and Optimization A-text on SYS003.	ILACBL50	Phase 50	LA
	Receives intermediate Procedure A-text and intermediate Optimization A-text, P2-text on SYS002; produces Procedure A-text on SYS001.	ILACBL51	Phase 51	MA
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.	ILACBL60	Phase 60	NA

Diagram 2. Part 6. Method of Operation: Object Module Production





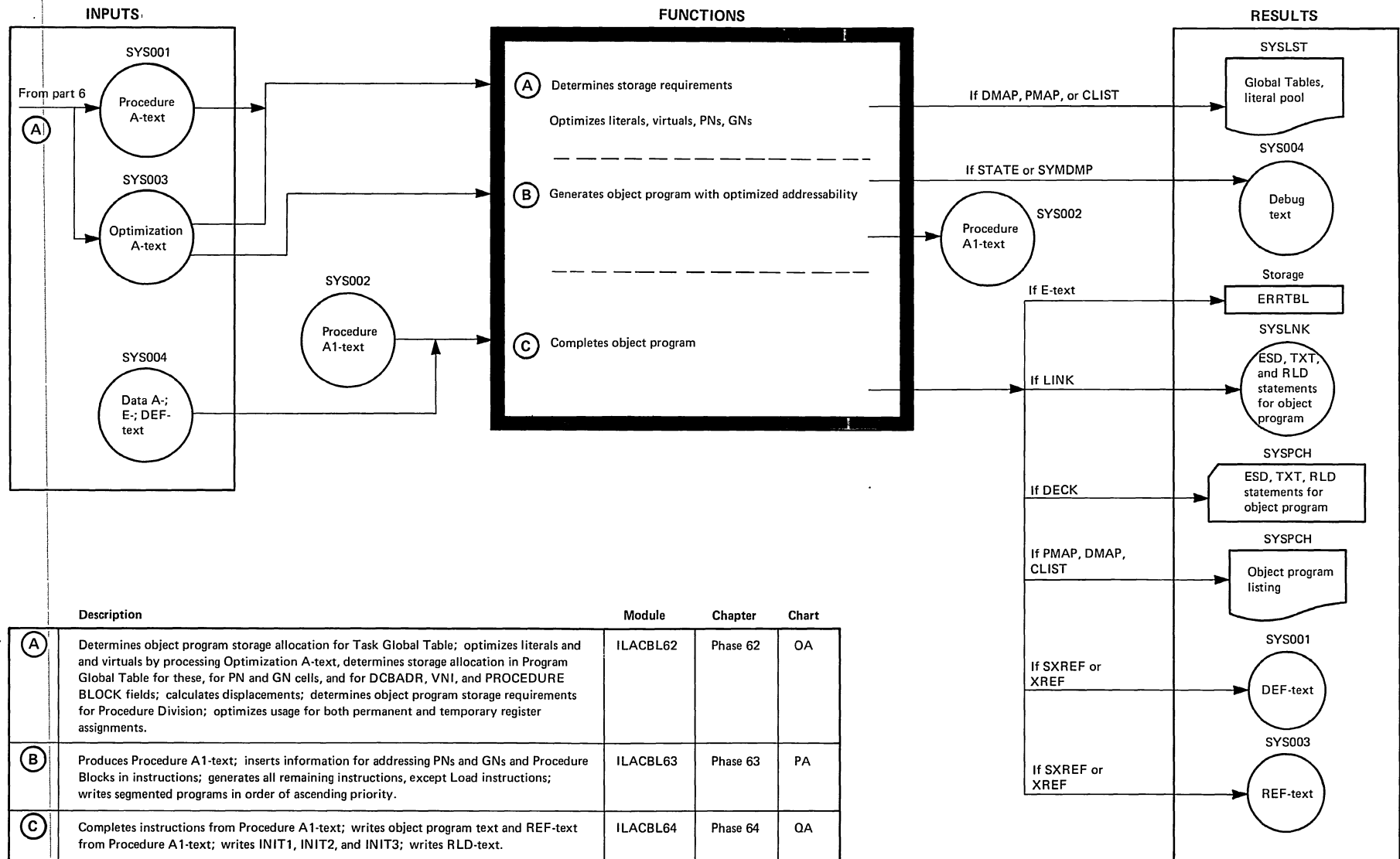
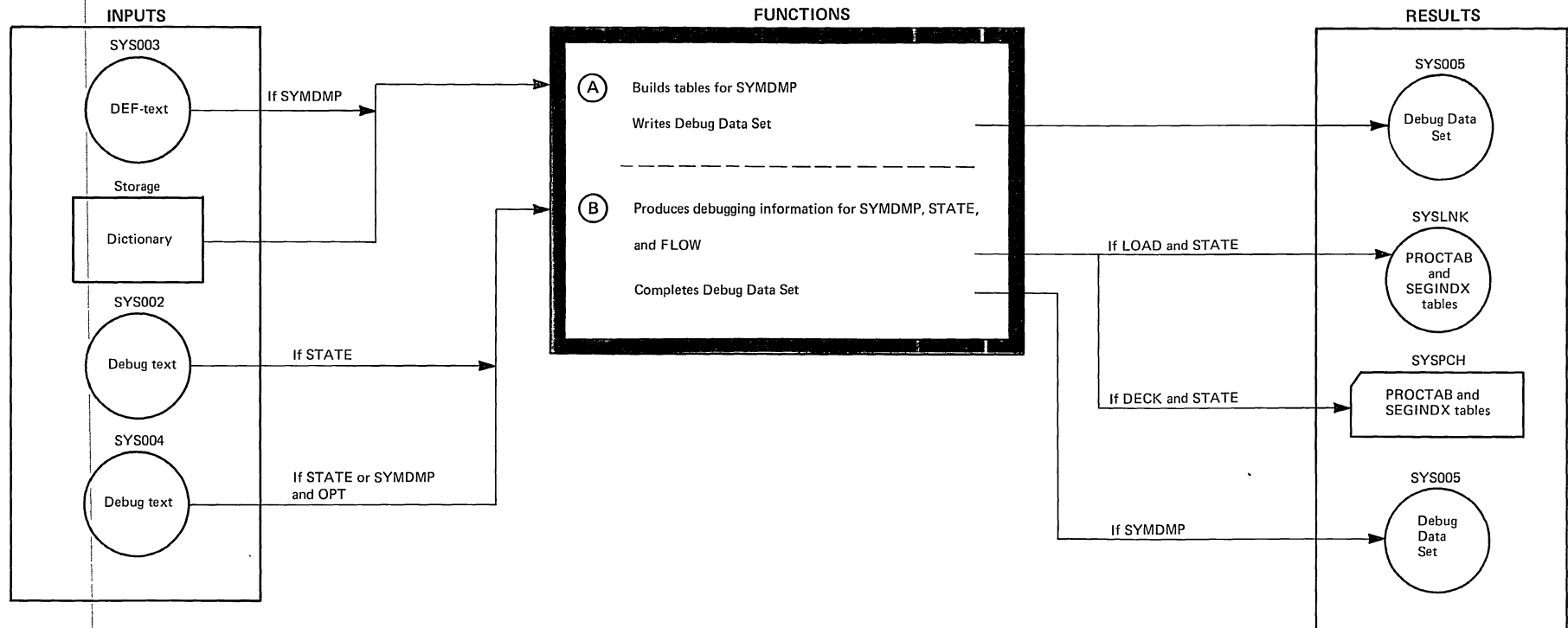


Diagram 2. Part 7. Method of Operation: Optimization of Object Module (Optional)

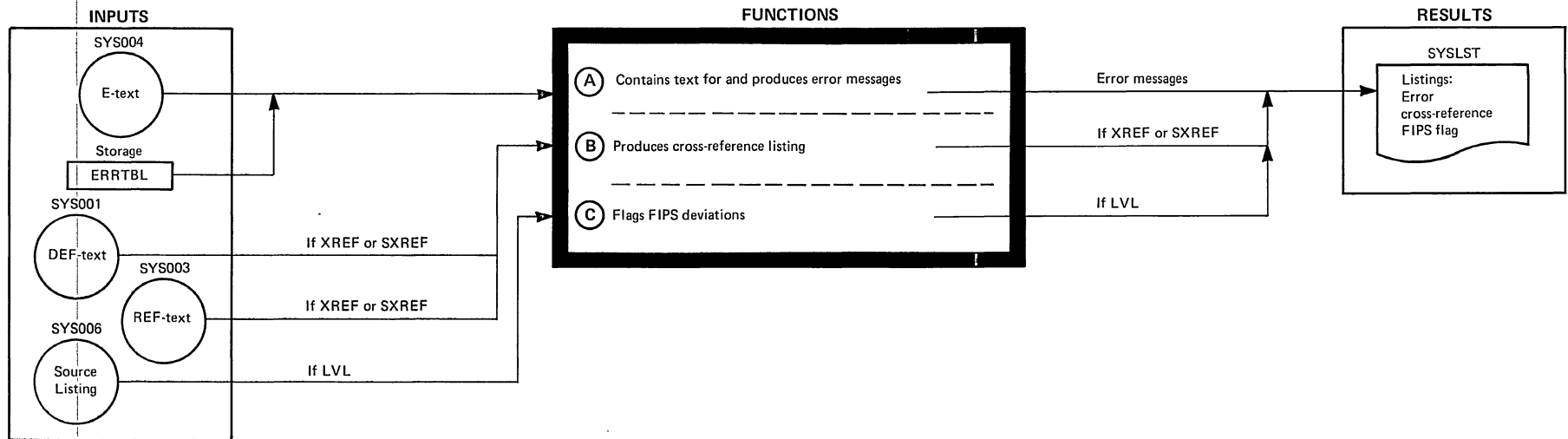




	Description	Module	Chapter	Chart
(A)	If SYMDMP is in effect, builds OBODOTAB table for OCCURS . . . DEPENDING ON clauses; builds DATATAB table; writes tables on SYSUT5 data set.	ILACBL25	Phase 25	IA
(B)	For the FLOW option, stores number of traces requested in DEBUG TABLE of Task Global Table. For STATE, 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).	ILACBL65	Phase 65	RA

Diagram 2. Part 8. Method of Operation: Debug Data Set Creation (Optional)





	Description	Module	Chapter	Chart
A	Produces E-text as it scans source program listing and analyzes syntax.	ILACBL10 ILACBL12 ILACBL11 ILACBL20 ILACBL22 ILACBL21 ILACBL25 ILACBL30 ILACBL40 ILACBL45 ILACBL50 ILACBL51	Phase 10 Phase 12 Phase 11 Phase 20 Phase 22 Phase 21 Phase 25 Phase 3 Phase 4 Phase 45 Phase 50 Phase 51	
	Contains text for error messages. Formats messages and prints them on SYSLST.	ILACBL70	Phase 70	
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.	ILACBL61	Phase 61	
C	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).	ILACBL80	Phase 80	

Diagram 2. Part 9. Method of Operation: Error Messages, Diagnostics, and Cross-Reference Listings



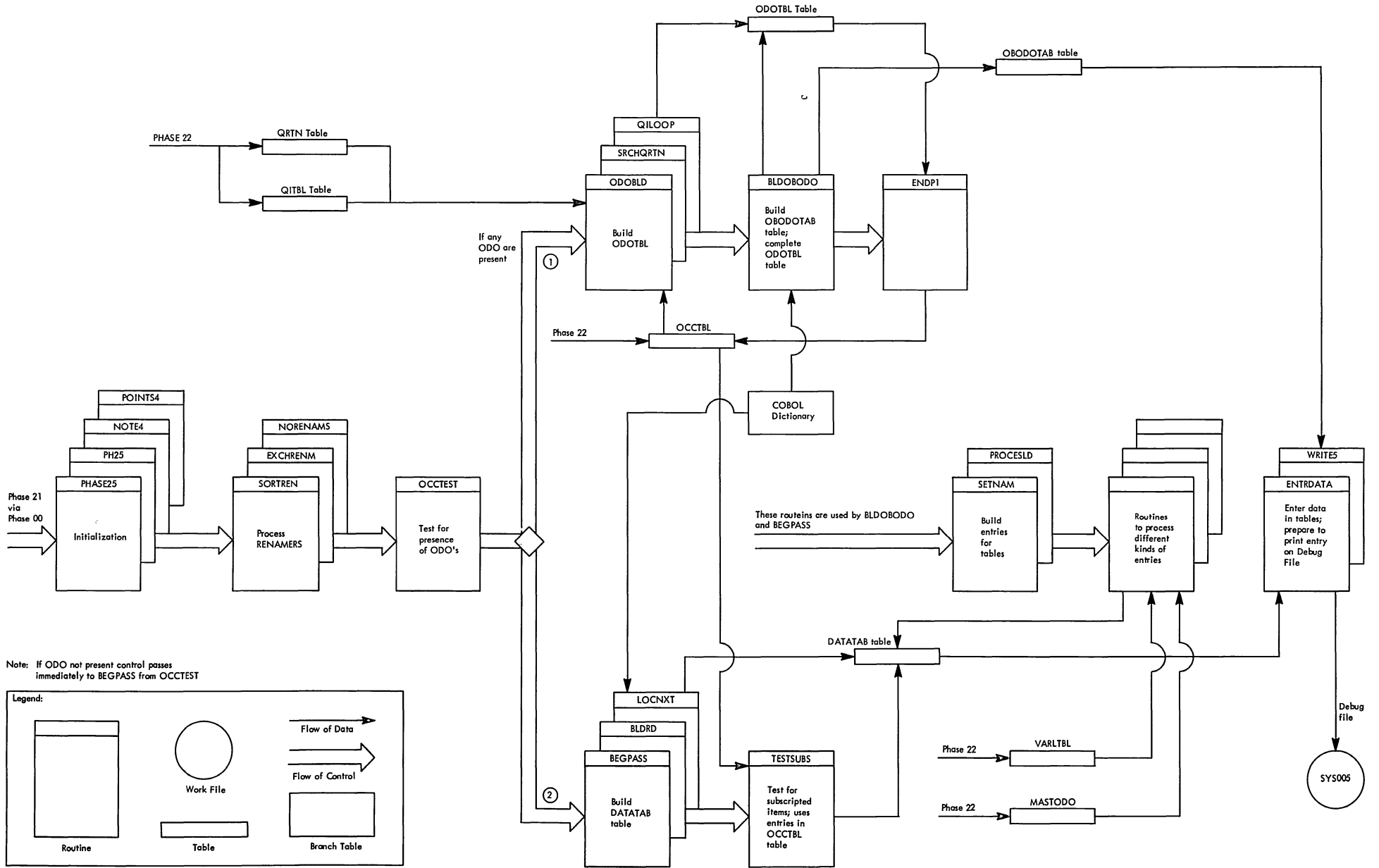


Diagram 3. Phase 25 Operations









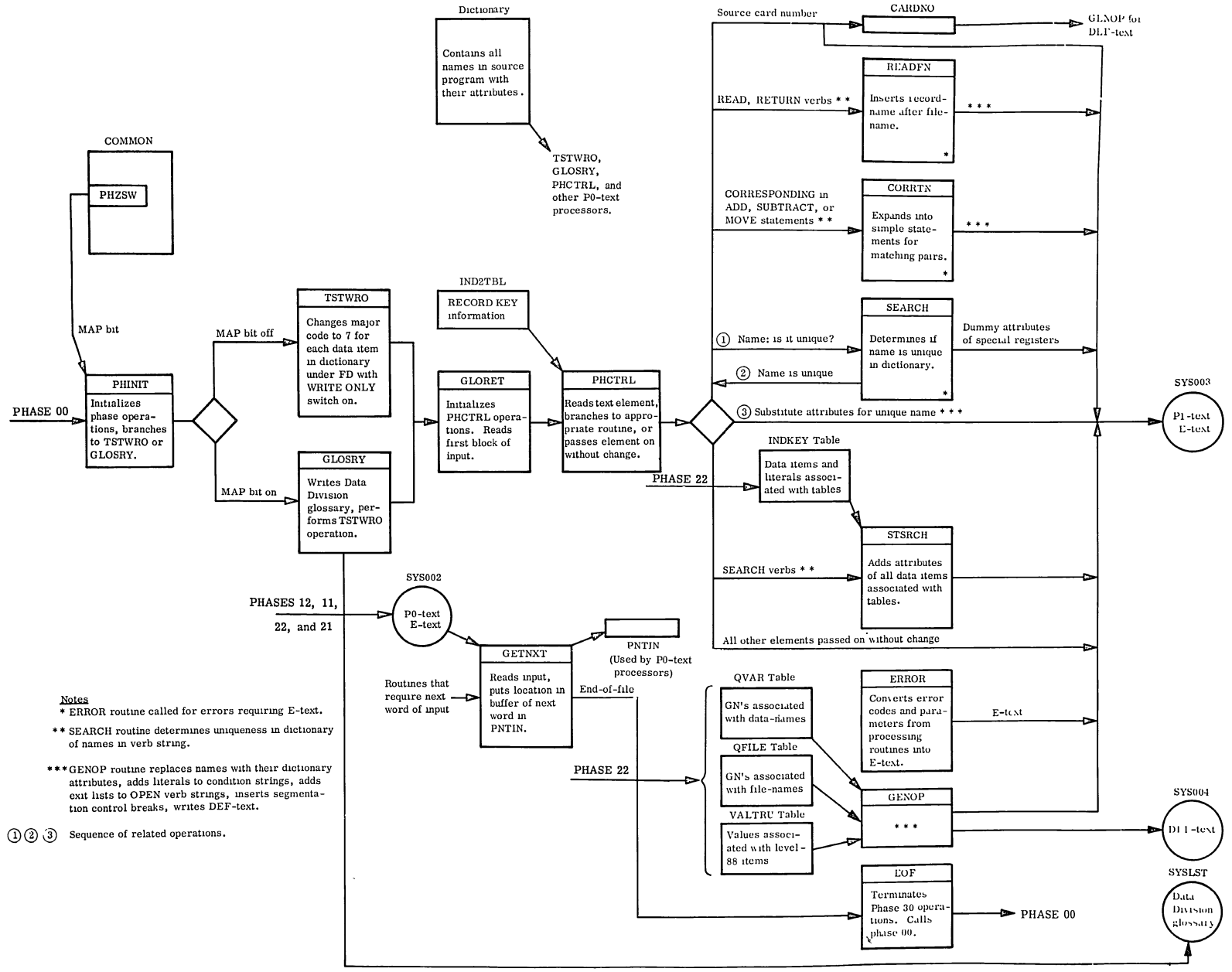


Diagram 5. Phase 30 Operations



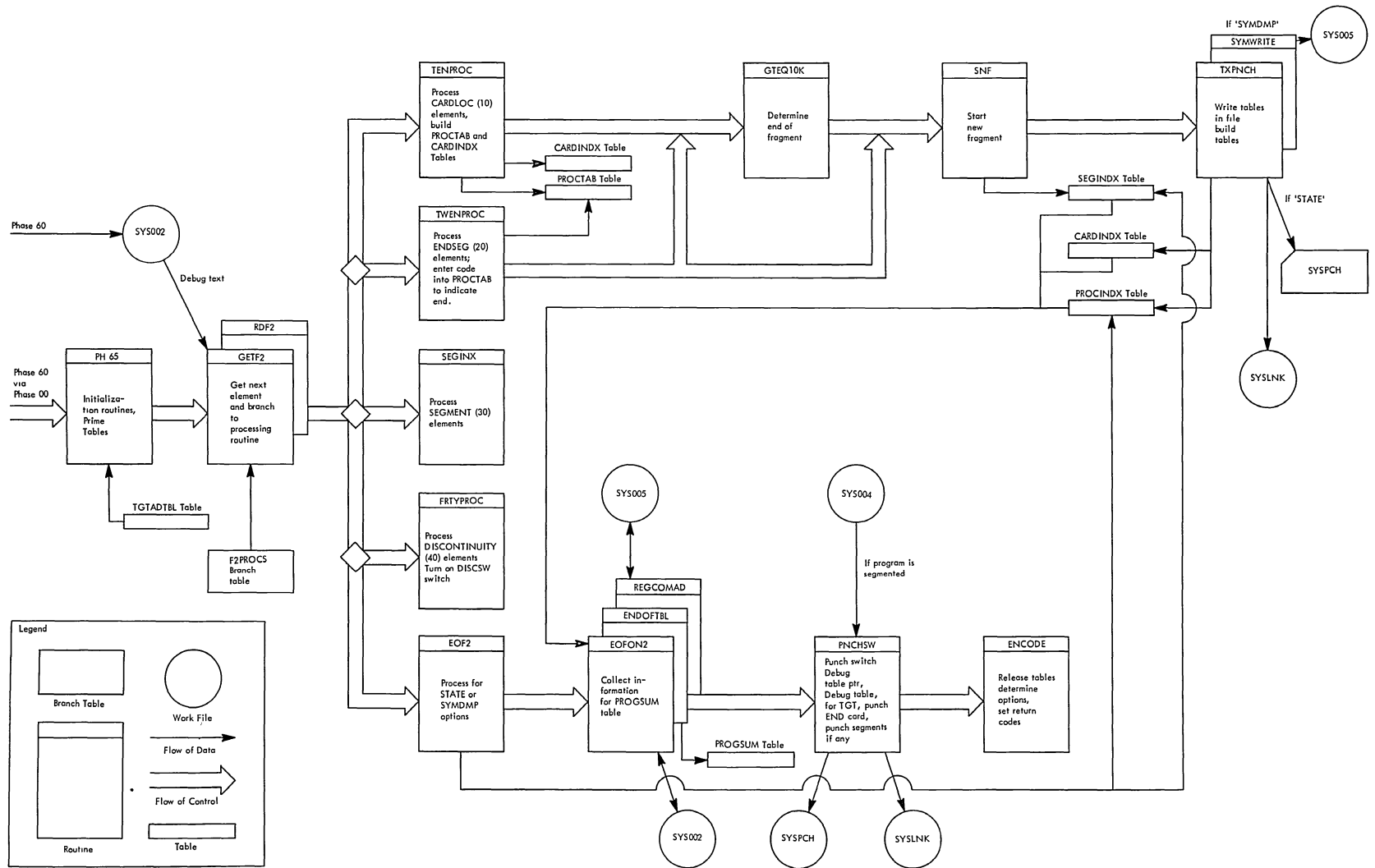


Diagram 6. Phase 65 Operations



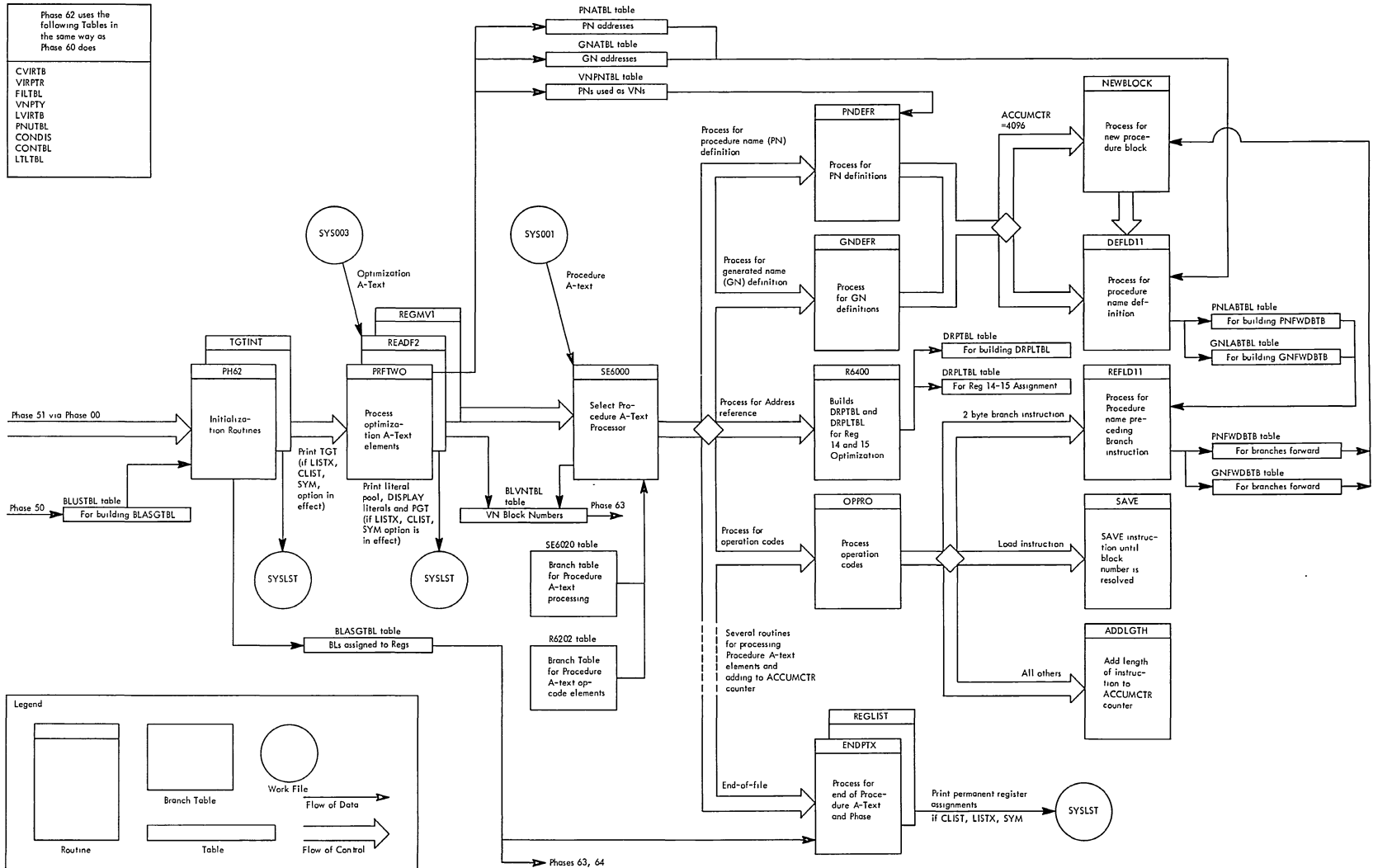


Diagram 7. Phase 62 Operations





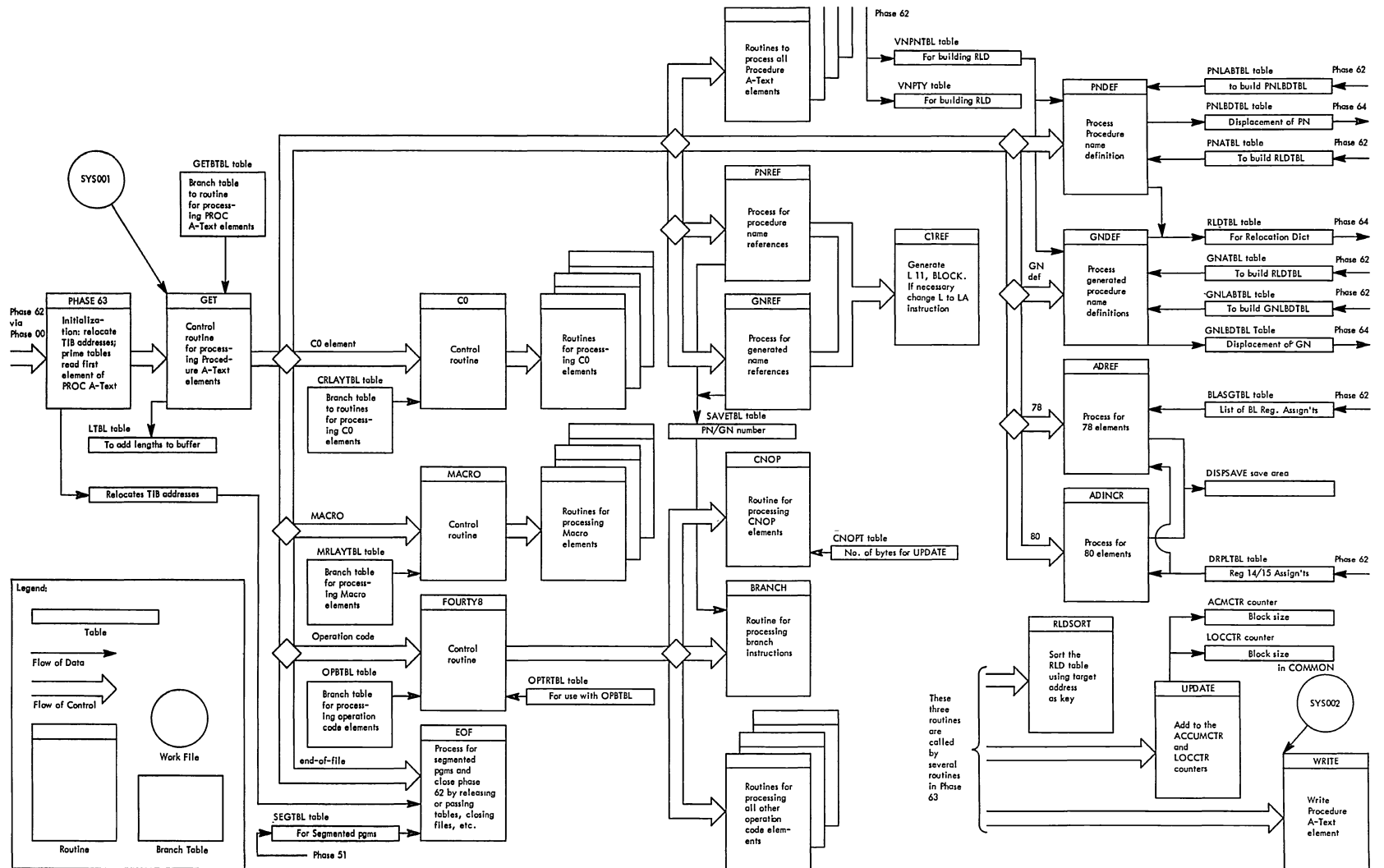


Diagram 8. Phase 63 Operations



INDEX

(Where more than one page reference is given, the major reference is first.)

- nnnn name cells 448
- A(INIT1) field (TGT) 432
- A-text generator
  - called by Arithmetic Translator 96
  - literals 99
  - parameter cells 100-103,99
  - virtuals 99
- ABEND 409
- abnormal termination during
  - compilation 409
- ABS.LIN cell 448
- ACCEPT statement
  - entries in SPNTBL table 40
  - output coding 468
  - phase 11 processing 49
  - phase 51 processing 106
- access method, effect on coding 106
- ACCESS Routine
  - ENTDEL 423
  - ENTNAM 422
  - ENTPTR 423
  - GETPTR 423,55
  - INTACC 422
  - LATACP 424
  - LATGRP 425
  - LATRNM 423
  - LATRPT 424
  - LDELNM 424
  - LOCNXT 424
  - TAMEIN 422
- ACCESS routines
  - definition 470,421
  - functions 422
  - handling CORRESPONDING 73
  - HASH table 421,73
  - initialization 422
  - list 422
  - phases involved 421
  - residence 421
  - sectional Procedure Division 50
- ACCESW cell (COMMON) 275
- ACCUMCTR counter 134
- ACMCTR counter 139
- ADATAB cell (COMMON) 281
- ADD CORRESPONDING option 74
- ADD verb 74
- address element 58
- address increment element 139
- address reference element 139
- addressing parameter field (see IDK field)
- ADSTAT cell (COMMON) 275
- AGETALL cell (COMMON) 279
- AHSEGMSG cell (COMMON) 286
- AINSRT cell (COMMON) 275
- ALS-ROUT routine
  - description 441
  - flowchart 250
- ALSTAM cell (COMMON) 275
- ALTER statement
  - execution diagram
  - flow of control 84
  - optimization (OPT) processing 130
  - phase 11 processing 49
  - phase 40 processing 82-84
  - PN cells in the PGT 117
- ALTER/GO TO pair 82-84
- AMAINF cell (COMMON) 275
- AMICTR cell (COMMON) 281
- AMILOC cell (COMMON) 282
- AMOVC cell (COMMON) 279
- APOST option 21
- appendix A. table and dictionary
  - handling 421-429
- appendix B. object module 430-439
- appendix C. report writer
  - subprogram 440-453
- appendix D. generated code for input/output
  - verbs 454-469
- APRIME cell (COMMON) 275
- arithmetic operator hierarchy 87
- arithmetic strings 87
- Arithmetic Translator 96-98
  - compile-time arithmetic 97
  - functions 96
  - general description 96
  - maximum operand size 97
  - register allocation 98
  - storage allocation 98
  - work area 97
- arithmetic verb strings 96
  - (see also Arithmetic Translator)
  - data-name operands 96
  - intermediate results 97
  - mode of operation 97
  - operand types 96
  - temporary results 97
- arithmetic verbs, register usage 98
- ASCTAB table
  - description 287
  - use in phase 21 68
- associated files
  - ASCTAB table 287
  - definition 470
  - device code for 343
  - organization code for 343
  - phase 21 processing 68
- AT END clause 89
- ATF-text
  - data-name DEF elements 58
  - definition 418
  - description 348
  - elementary item entries 53
  - format 52
  - group routine entries 53
  - OCCURS...DEPENDING ON clauses 58
  - use in phase 22 58
- ATFTXT work area 53
- ATPLENT cell (COMMON) 286

base locators  
 assignment in phase 22 57  
 BL number  
 assignment 57  
 base register assigning 122  
 definition 470  
 FDS 58  
 locating data 419  
 locating RWS elements 449  
 OPT processing 130-133  
 subscripts 94  
 Working-Storage Section 58  
 BLL number  
 assignment 57  
 definition 470  
 Linkage Section 58  
 example of OPT processing 133  
 optimization (OPT) processing 130-133  
 SBL number  
 assignment 57  
 definition 472  
 variably located fields 59  
 types 57  
 use by phase 60 57  
 use by phase 62 130-133  
 use of permanent registers 57  
 base registers 122  
 BASIS library  
 action if not found 37  
 action in phase 10 37  
 invalid name 34  
 BCB (Buffer Control Block) 462  
 BEGIN routine 52  
 BEGINPH routine 60  
 BEGPASS routine 69  
 BL CELLS field (TGT) 432  
 locating RWS elements 449  
 space allocation 113  
 BL number (see base locators)  
 BLASGTBL table, description 287  
 BLCTR cell (COMMON) 280  
 addressing parameter 57  
 assigning BL number 57  
 processing FDS 58  
 TGT space allocation 113  
 BLDOBODO routine 69  
 BLKCTNS routine 60  
 BLL CELLS field (TGT)  
 description 432  
 space allocation 113  
 BLL number  
 assignment 57  
 definition 470  
 Linkage Section 58  
 BLLCTR cell (COMMON) 277  
 assigning BLL number 57  
 description 277  
 TGT space allocation 94  
 BLOCK CONTAINS clause 60,61  
 blocking factor, effect on coding 106  
 BLTABL table  
 buffer generation 68  
 description 287  
 BLUSTBL table  
 description 289  
 primed by phase 50 103  
 BLVNTBL table 289  
 BMBSRN routine 53  
 branch instructions  
 phase 62 processing 134  
 phase 63 processing 138  
 BSUBRN routine 53  
 BUF option 21,37  
 BUFCNLS area 410  
 BUFCNLS cell 37  
 Buffer Control Block (BCB) 410  
 buffer generation, use of BLTABL table 68  
 buffers  
 address of area 35  
 allocation of space 35-37  
 assignment 36  
 beginning of area 68  
 finding contents 350  
 generating 68  
 locating 36  
 pointer in POINT table 28  
 size determination 35  
 size of area 68  
 BUFGEN routine 68  
 BUFSIZ cell (COMMON) 282  
 BUFTAB table  
 buffer generation 68  
 description 289  
 BUSAGE routine 53  
 CALL statement  
 allocating PGT space 116  
 CALL string 82  
 CANCEL request 27  
 card numbers  
 generated by phase 11 48  
 object text listing 122  
 CARDINDX table  
 built in phase 65 127  
 description 290  
 CARDLOC elements  
 (see also Debug-text)  
 phase 65 processing 126  
 use in building debug file 126  
 CARDNO cell 81  
 carriage control word 40  
 CATAL option 21,111  
 CATALR option 21  
 CBL card  
 information passed to phase 00 37  
 OPT option 129  
 options permitted 21-23  
 phase 01 actions 18  
 STXIT option 106  
 SYMDMP option 69  
 CBNOFND routine 27  
 CCBLOC cell (COMMON) 282  
 CE abstract worksheet 420.3  
 CFF-ROUT routine  
 description 446  
 flowchart 252  
 CHECKPT CTR field (TGT)  
 description 432  
 space allocation 113  
 CHF-ROUT routine 446  
 CHKMODE routine 61  
 CKPCTR cell (COMMON)  
 description 277  
 TGT space allocation 113

CKPTBL table	COMMON, cells (continued)
description 291	BLICTR 277
entries for RERUN 41	BUFSIZE 282
clause compatibility	CCBLOC 282
BLOCK CONTAINS clause 60	CKPCTR 277
DTF generation 60	COMMAD 279
CLIST option	CORESIZE 281
definition 21	COS 275
effect on PGT listing 116	CRDNUMXX 282
phase 60 processing 111,118	CURCRD 282
phase 64 processing 141	CURSGN 279
CLOSE macro instruction 28,106	DATABDSP 279
CLOSE REEL statement, output coding 459	DATATBNM 281
CLOSE statement	DATE 282
output coding 457	DCPTR 278
phase 51 processing 106	DICADR 278
closing files 28	DICND1 277
CNTLTBL table 291	DICND2 277
CNTRL macro instruction 106	DICPTR 278
COBOL bits in DTFs 66	DLSVAL 278
COBOL Library Subroutines	DTFNOXX 282
(see also COBOL subroutines)	DTFNUM 282
changing operating mode 96	DNCNT 282
definition 470	ERF4SW 278
example of linkage generating 108	ERRSEV 278
for VSAM 110	FIL5BUF 281
READ processing 106	FLOWSZ 279
SEARCH ALL statement 89	FSTCDNM1 282
USE coding 106	GNCTR 276
STATE processing (see STATE option)	GTING 281
SYMDMP processing (see SYMDMP option)	IDENTL 279
WRITE processing 106	INDEX 281
COBOL space	INDEX1 279
assigning 426	INTVIRT 282
definition 470	IOPTRCTR 279
COBOL Subroutines	IPRECS 286
(see also COBOL Library Subroutines)	LABELS 276
ACCEPT	LCSECT 278
ACCEPT coding 106	LISTERSW 282
DISPLAY	LNCOUNT 286
DISPLAY processing 109	LOCCTR 275
example of linkage codes 109	LOCTMCTT 282
generating call 108	LTLCTR 276
for VSAM 110	NODECTR 281
segmentation	OBODOTBN 281
generating call 108	ODOCTR 277
COBOL word list 355-358	ONCTR 280
COBOLRWD bit in DTFs 66	PARMAX 281
CODE-cell cell 447	PFMCTR 280
coding organization, phase 30 71	PHZSW 284
COM-REG register 370	PHZSW1 284
COMMAD cell (COMMON) 279	PHZSW2 284
COMMON 275-286	PHZSW3 284
cells	PHZSW4 285
ACCESW 275	PH1BYTE 285
ADATAB 281	PH6ERR 280
ADSTAT 275	PH7LOD 282
AGETALL 279	PMAPADR 282
AHSEGMSG 286	PNCTR 276
AINSRT 275	PRBLDISP 276
ALSTAM 275	PRBLNUM 281
AMAINF 275	PROCCTR 281
AMICTR 281	PROGID 276
AMILOC 282	PSVCTR 280
AMOVDC 279	PTYNO 278
APRIME 275	RELADD 275
ATPLENT 286	RELLOC 281
BLCTR 280	RGNCTR 278

COMMON, cells (continued)

RPNCNTR 279  
 RPTSAV 278  
 SA2CTR 278  
 SBLCTR 277  
 SDSIZ 279  
 SDTFCTR 282  
 SEGLMT 279  
 SEQERR 277  
 SPACING 281  
 SRTRERUN 286  
 SUBCTR 281  
 SWITCH 283  
 SWITCH1X 285  
 SWITV2 282  
 TAMNAD 275  
 TIBs 275  
 TIME 282  
 TMCNTBSZ 282  
 TSMAX 276  
 TS2MAX 277  
 TS3MAX 279  
 TS4MAX 279  
 VIRCTR 276  
 VLCCTR 277  
 VNCTR 280  
 VNILOC 281  
 VNLOC 281  
 VRBCNT 282  
 WCMAX 276  
 WSDEF 278  
 XSACTR 280  
 XSWCTR 280  
 contents 275-286,34  
 counters used to allocate space 133  
 definition 470  
 general concept 275  
 option flags 35  
 residence 34  
 TGT-field counters 112,113  
 Communications Area (see COMMON)  
 Communications Region (see COMMON)  
 compare in System/370 109  
 compiler  
   active phase 24  
     (see also LINKCNT)  
   buffers  
     assignments 36  
     contents 410  
     size 35,37  
   code organization 66  
   communications area (see COMMON)  
   concept 14  
   design of 14  
   diagnostic aids 409-420  
   execution-time register use 411  
   file handling 29-34  
   initialization 35  
   input (see input)  
   layout of storage 13  
   Linkage Editor Map 265-274  
   minimum buffer size 35  
   operational considerations  
   options 20-23  
   output (see output)  
   overall design 18-20,14  
   overview 477-480  
   parameters 20-22

phases  
   addresses 409-410  
   approximate sizes 13  
   intercommunication 14  
   physical characteristics 13  
   relationship to DOS/VS system 13  
   storage layout 13  
   storage locations 13  
     names 13  
     order of loading 13  
     overlying in storage 13  
     translation 14  
     use of work files 29-34  
 compiler buffers  
   assignments 36  
   contents 410  
   size 35,37  
 compiler options (see options)  
 compiler tables  
   allocation of space 37  
   current record 410  
   current status 410  
   exceeding permissible size 34  
   for Q-routines 58  
   list, by phases 263-264  
   locating in storage 410  
   location in storage 421  
   Main Free Area 421  
   maximum length 426  
   object module map 430  
   passed by phase 22 59  
   permissible size 35  
   processing in phase 00 35,429  
   Report Writer feature use 45  
   RWS data items 449  
   RWS routines 449  
   saved register contents 418,411-416  
   SDTF location 461  
   table area 421  
   table contents 410  
   table size 426  
 compiler texts (see internal texts)  
 COMPUT routine 87  
 COMPUTE statement 87  
   EVAL string 88  
   ON SIZE ERROR option 88  
   phase 40 processing 87  
   ROUNDED clause 88  
 COMWRK work area 39  
 CONDIS table  
   description 291  
   optimizing literals 115  
   PGT space allocation 118  
   segmented program object text 121  
 condition-names 78  
   UPSI feature 58  
 condition-string  
   P1-text 79,80  
   with VALUE THRU clause 80  
   without VALUE THRU clause 79  
 Configuration Section 40  
 Constant A-text 99  
 CONTBL table  
   description 292  
   optimizing literals 115  
   PGT space allocation 118  
   segmented program object text 122  
 control and input/output diagram 485

Control breaks 104  
 Control-field save-area names 448  
 COPY library  
     flag in Data IC-text 53  
     phase 10 actions 39  
 COPYPCH option 23  
 COPYRN routine 53  
 CORESIZE cell (COMMON) 281  
 CORRESPONDING option 73  
 CORRTRN routine 73  
 COS cell (COMMON) 275  
 COS routine  
     address location 24  
     handling segmentation 34  
     processing between phases 28  
 COSPARM area 462  
 COUNT CHAIN ADDRESS field (TGT) 432  
 COUNT LINKAGE AREA field (PGT) 435  
 COUNT option  
     definition 21  
     phase 40 processing 81  
     virtuals required 99  
 COUNT table 437,470  
 COUNT TABLE ADDRESS field (TGT) 432  
 counters in COMMON 113  
 CRDNUMXX cell (COMMON) 282  
 critical program break, action by  
     phase 20 53  
 cross-reference list  
     (see also Lister option)  
     diagram 497  
     locating RWS elements 449  
     phase 61 processing 144,145  
     use of REF-text 118  
     used by phase 11 48  
     used in phase 20 52  
     used in phase 22 55  
 CSYNTAX option  
     (see also SYNTAX)  
     definition 23  
     phase 22 processing 59  
     phase 30 processing 78  
     phase 40 processing 91  
 CTB-ROUT routine  
     description 441  
     flowchart 242  
 CTF-ROUT routine  
     description 446  
     flowchart 253  
 CTH-ROUT routine 446  
 CTL.CHR cell 447  
 CTL.LVL cell 447  
 CTLTBL table 292  
 CURCRD cell (COMMON)  
     description 282  
     locating processing record 410  
     use in phase 20 52  
     use in phase 22 55  
 CURGCN cell 39  
 CURRENCY-SIGN clause 40  
 CURRENT-DATE register 370  
 CURRENT PRIORITY field (TGT) 432  
     set by phase 65 112  
 CURSGN cell (COMMON)  
     contents 40  
     description 279  
 CVIRTB table  
     description 293

optimizing virtuals 116  
 segmented program object text 122  
 virtual allocation 116  
  
 "d" addressing parameter 60  
 Data A-text  
     associated file DTFs 67  
     data-name DEF elements 58  
     data-names and file-names 41  
     definition 470  
     description 349-350  
     location on file SYS004 123  
     phase 22 generation 58  
     phase 60 processing 123  
     purpose 123  
     Q-routine identification elements 58  
     VALUE clause constants 58  
     Working-Storage Section address  
         elements 58  
 data areas 275-408  
 Data Division  
     addressing with OPT 132  
     controlling processing 39  
     processing by phase 10 41  
     processing overview 18,477-480  
     syntax analysis 43  
     translation, diagram 487  
 Data IC-text  
     BCD names 53  
     copied unchanged by phase 20 52  
     data-name DEF elements 58  
     data-names and file names 42  
     definition 470  
     description 341-347  
     FD element 343  
     FD entries 40  
     FD processing in phase 10 42  
     FILE-CONTROL paragraph 40  
     input to phase 20 52  
     items from Environment Division 40  
     LD element 341  
     processing in phase 20 53  
     processing in phase 21 60  
     RD element 343  
     RD processing phase 10 42  
     read by phase 20 53  
     Report Writer data items 47  
     SD element 343  
     SD processing in phase 10 42  
     use in phase 22 58  
 data-names  
     discussion 78  
     location in storage 419  
     REF-text elements 144  
     reference elements 78  
     subscripts 94,95  
 DATABDSP cell (COMMON) 279  
 DATATAB table  
     description 398  
     phase 25 processing 69  
     phase 65 processing 128  
 DATATBL table 293  
 DATATBNM cell (COMMON) 281  
 DATE cell (COMMON)  
     description 282  
     setting in phase 01 37,35  
 DATE-COMPILED paragraph 40  
 DBGTL table

description 293  
 generating P2-text 82  
 processing verb strings 81  
 DCPTR cell (COMMON) 278  
 DDSCN routine 42  
 debug data set, creation of, diagram 495  
 Debug File 395-405  
   built by phase 25 69  
   CARDINDX table 395  
     built by phase 65 127  
   DATATAB table 398  
     built by phase 25 69  
   information about 37  
   OBODOTAB table 397  
     built by phase 25 69  
   phase 65 processing 127  
   PROCINDX table 405  
     built by phase 65 127  
   PROCTAB table 404  
     built by phase 65 126  
   PROGSUM table 396  
     built by phase 65 127  
   SEGINDX table 405  
     built by phase 65 127  
   tables built by phase 22 59  
 debug file tables 395  
 DEBUG LINKAGE AREA field (PGT) 435  
 debug options (see FLOW option; STATE  
   option; SYMDMP option)  
 DEBUG TABLE field (TGT)  
   description 342  
   phase 65 processing 126  
 DEBUG TABLE POINTER field (TGT) 342  
   set by phase 65 112  
 Debug-text  
   description 384  
   phase 65 processing 126  
   use in building Debug File 127,126  
 DEBUG verb processing  
   phase 11 processing 49  
   phase 40 processing 82  
 debugging compiler (diagnostic  
   aids) 409-420  
 debugging packet 82  
 DECIMAL-POINT clause 40  
 DECK option  
   definition 21  
   effect on buffers 37  
   phase 60 processing 111  
   phase 64 processing 141  
 DECK option in LST card 24  
 declaratives  
   effect on coding 106  
   OPEN verb 49  
 Declaratives Section  
   processing in phase 11 50  
 DEF-text  
   data-name DEF elements 58  
   definition 470  
   description 383  
   from Procedure Division 19  
   generation in phase 30 72  
   location on file SYS004 123  
   processing by phase 60 123  
   processing in phase 61 144-145  
 DEFSBS table  
   description 294  
   MOVE statement 82  
 DELETE verb processing 49  
 delimiter 470  
 delimiter pointer  
   definition 470  
   determining 41  
   group items 55  
 DEPENDING ON option 60  
 DESTROY macro element 132  
 DET-ROUT routine  
   description 447  
   flowchart 240  
 DETTBL table 294  
 device type, effect on coding 105  
 DFREER routine 98  
 diagnostic aids 409-420  
   abnormal termination during  
     compilation 409  
   CE abstract worksheet 420.3  
   dump produced 409  
   erroneous compiler output 418  
   error message listing 409,420  
   input/output error messages 418  
   linkage editor map example 420  
   location of information in  
     storage 409-410,419  
   register saving 418  
   register usage 411-417,419-420  
   storage layout 418  
   tables used 410  
 diagnostic assistance (CE) 420  
 diagnostics 146-148  
 diagrams, foldout 475-509  
 DICADR cell (COMMON) 278  
 DICND1 cell (COMMON)  
   description 277  
   use by phase 30 72  
 DICND2 cell (COMMON)  
   description 277  
   use by phase 30 72  
 DICOT table  
   description 295  
   use in phase 25 69  
 DICPTR cell (COMMON) 278  
 DICSPC routine 427  
 DICTBD routine  
   description 57  
   elementary item processing 53  
   purpose 55  
 Dictionary  
   addressing parameters (see IDK field)  
   area 421,422  
   assigning space 426  
   building entries in phase 22 55-58  
   completing entries in phase 22 57,55  
   data-name entries 42  
   Declaratives Section entries 50  
   definition 470  
   description 385-394,421-422  
   dummy entries 55  
     UPSI feature 40  
   entries made by phase 22 55  
   FD entries, filling in reserved  
     space 60  
   FD entry 386  
   file attributes 61  
   File Section entries 57  
   file-name entries 42  
   formats of entries 385-394



generating P1-text 81  
 glossary production 71  
 handling, table and dictionary 421-429  
 handling spill 28  
 incomplete index-name entries 55  
 index-name entry 389  
 interaction with tables 50  
 LD entry 388  
 Linkage Section entries 58  
 locating entries 421-422  
 logical position in phase 22 55  
 major code 385-389  
 minor code 388,390  
 organization 421  
 PERFORM attributes 85  
 phase 21 functions 60  
 pointer 78  
 pointer for WHEN statement 90  
 preprocessing in phase 22 55  
 procedure-name characteristics 50  
 procedure-name entries 48,50  
 procedure-name (paragraph) entry 385  
 procedure-name (section) entry 385  
 procedure-names 78  
 Q-routine entries 58  
 RD entry 388  
 replacing names 78  
 Report Section entries 58  
 residence 421  
 SD entry 387,60  
 SEARCH processing 74  
 searching sectioned Procedure  
   Division 50  
 skeleton FD entry 55  
 space allocation 35  
 spill onto disk file 422  
 SYMDMP processing in phase 25 69  
 Working-Storage Section entries 57  
 Dictionary Area 421  
 Dictionary attributes  
   condition-name 78  
   definition 470  
   Q-routine bit 78  
   replacing names 72  
   substituting for names 78  
 Dictionary entry formats 385-394  
 Dictionary handling (see ACCESS routines)  
 Dictionary pointer  
   definition 471  
   use after phase 30 78  
   use in building OBODOTAB table 501  
 Dictionary section addresses 421  
 Direct A-text 99  
 DIRECTOR routine 55  
 directory 259-274  
 DISCONTINUITY element  
   (see also Debug-text)  
   phase 65 processing 126  
   use in building Debug File 127  
 Diskette unit device 65  
 DISPLAY literals, optimization for PGT 115  
 DISPLAY LITERALS field (PGT)  
   description 436  
   optimizing with OPT 130  
   space allocation 118  
 DISPLAY statement  
   entries in SPNTBL table 40  
   output coding 468  
   phase 11 processing 49  
   phase 51 processing 106  
   subroutine linkage 108  
 DLSVAL cell (COMMON) 278  
 DNCNT cell (COMMON) 282  
 DNTOR1 routine 108  
 DOP work areas 92  
 DOS/VS system  
   interface with compiler 24  
   relationship with compiler 13  
   returning control 28  
   supervisor 418  
   UPSI-feature-names 58  
   use of Control Program 13  
 double buffering, effect on coding 105  
 drop tables (see DRPTBL table; DRPLTBL  
   table)  
 DRPTBL table  
   description 295  
   use in register assignment 132  
 DRPLTBL table  
   description 295  
   use in register assignment 132  
 DSPLAC routine 112  
 DTF  
   address element 67  
   associated files 67  
   choice of type 61  
   COBOL bits 66  
   COBOL indicators 66  
   COBOLRWD bit 66  
   common parameters 60  
   constant definition elements 67  
   DTFCD, number of DTFs 62  
   DTFDA  
     number of DTFs 62  
     pre-DTF 62,63-64  
   DTFDU 65,62  
   DTFIS  
     number of DTFs 62  
     pre-DTF 65  
   DTFMT  
     number of DTFs 62  
     pre-DTF 62,63  
   DTFPR, number of DTFs 62  
   DTFSD  
     number of DTFs 62  
     pre-DTF 62,63  
   generating 60-66  
   IOAREA address 68  
   location in object module 430  
   location in storage 419  
   number assigned 62  
   number required 62  
   order of addresses 419  
   pointer location 105  
   pre-DTF area 62  
   REWIND bit 66  
   DTF generation 60-66  
   clause compatibility checking 60  
 DTF generator  
   functions 61-66  
   GENDTFCD 61  
   GENDTFDA 61  
   GENDTFDU 61  
   GENDTFIS 61  
   GENDTFMT 61  
   GENDTFPR 61

- GENDTFSD 61
  - selecting 61
- DTF number
  - files with OCCURS...DEPENDING ON clauses 58
  - OBJSUB table element 58
  - QFILE table elements 58
- DTF parameters
  - Record form 60
  - Record size 60
- DTFADR CELLS field (TGT)
  - description 432
  - locating DTFs 418
  - space allocation 113
- DTFCDD 62
- DTFDA
  - number of DTFs 62
  - pre-DTF 62,63-64
- DTFDU
  - number of DTFs 62
  - pre-DTF 65
- DTFIS
  - number of DTFs 62
  - pre-DTF 65
- DTFMT
  - number of DTFs 62
  - pre-DTF 62,63
- DTFNOXX cell (COMMON) 282
- DTFNUM cell (COMMON)
  - description 282
  - TGT space allocation 113
- DTFPR 62
- DTFSD
  - number of DTFs 62
  - pre-DTF 62,63
- dummy entry for phase 80 50
- dump, producing for disaster-level messages 409
- dynamic dump request
  - (see also SYMDMP option)
  - phase 25 processing 69
  - phase 65 processing 126
- E.nnnn name cells 447
- E-point name cells 448
- E-text
  - definition 471
  - description 382
  - from Procedure Division 19
  - identification prefix 90
  - input to phase 20 52
  - input to phase 30 71
  - location on file SYS004 123
  - phase 21 processing 60
  - phase 30 processing 78
  - phase 51 processing 104
  - phase 60 processing 123
  - phase 70 processing 146
  - producing 19
  - RECFORM parameter 61
  - WHEN clause 90
  - written by phase 12 47
- EACTBL table 146
- EACT00 area 147
- elementary report items 47
- ELIPR routine 58
- ENDSEG elements
  - (see also Debug-text)
- phase 65 processing 126
  - use in building Debug File 127
- ENTDEL routine 423
- ENTDIC routine 61
- ENTNAM routine 422
- ENTPTR routine 423
- ENTRDATA routine 69
- ENTRY-SAVE field (TGT) 432
- Environment Division
  - controlling processing 40
  - processing 40
  - processing overview 18
  - syntax analysis 43
  - translation, diagram 487
- ENVSCN routine 40
- ENVTBL table
  - APPLY clause switch 41
  - CKPTBL bit 41
  - description 296-298
  - FNTBL table use 42
  - MULTIPLE FILE TAPE switch 41
  - phase 10 processing 40-42
  - pointers 40
  - POSITION INTEGER field 41
  - SAME clause files 41
  - SELECT clause entries 40
- EOF routine 92
- EOJ macro instruction 24
- EQUATE string
  - built by phase 51 105
  - used by phase 60 112
- ERF4SW cell (COMMON) 278
- ERMSG area 78
- erroneous compiler output 418
- error conditions
  - (see also error messages; error processing)
  - ATF-text 59
  - background size 37
  - BUF parameter 37
  - checking by phase 20 52
  - clause compatibility 52
  - CORRESPONDING operands 73
  - Data IC-text 58
  - Dictionary requirements 72
  - E-text 382
  - EBCDIC key-names 58
  - execution-time messages 418
  - file-names after READ 72
  - file-names after RETURN 72
  - footnotes in phase 08 38.4
  - I/O error messages 418
  - invalid diagnostics 409
  - messages from phase 01 37
  - OCCURS...DEPENDING ON clauses 58
  - phase 05 processing 38.1
  - phase 08 processing 38.4
  - phase 22 processing 59
  - phase 30 messages 78
  - phase 30 processing 78
  - phase 70 processing 146-147
  - premature EOF 37
  - producing a dump for disaster level messages 409
  - SEARCH verb 75
  - syntax checking
    - in phase 05 38.1
    - in phase 12 47

TAMER-detected 34  
 unopened file 37  
 unrecoverable 34  
 using table SRCHKY 59  
 using table VALTRU 59  
 error message listing, obtaining 147  
 error message parameters 146  
 error messages  
   (see also error conditions; error processing)  
   diagram 497  
   disaster level 409  
   execution-time 418  
     (see also phase 70)  
   FIPS flagging 148  
   identification code 420  
   input/output 418  
   invalid 420  
   listing 147  
   parameters 146  
 error processing  
   (see also error conditions; error messages)  
   COBOL word checking 91  
   E-text severity code 104  
   operand-checking 91  
   phase 40 91  
   severity code 104  
   unit-record error 106  
   verb checking 91  
   WHEN conditions 90  
 ERROR routine, phase 30 79  
 ERROR routine, phase 40 91  
 errors detected by TAMER 34  
 ERRPRO routine 104  
 ERRS option 21  
 ERRSEV cell (COMMON)  
   description 278  
   testing for SUPMAP 111  
   use by phase 51 112  
 ERRTBL table  
   description 299  
   saving E-text 146  
   spilling 146  
 ESD (External Symbol Dictionary) 114  
 EVAL string 88  
 execution-time statistics (see COUNT option)  
 EXHIBIT statement 49  
 EXIT verb, phase 11 processing 49  
 EXIT5 routine 92  
 External Symbol Dictionary (ESD) 114  
  
 FD (see file description entries)  
 FD element  
   (see also file description entries)  
   Data IC-text 341  
   Dictionary entry 386,387  
 FDTAB table  
   description 299  
   phases 22 and 21 59  
 federal information processing standard (FIPS)  
   diagnostic messages 148  
   flagging 38  
   LVL option 22  
   phase 01 processing 38  
   phase 80 processing 148  
   scanning source program 148  
   SYS006 22,148  
 FIB (VSAM file information block) 68  
 FILE-CONTROL paragraph 40  
 file description entries (FDs)  
   assignment of BL number 57  
   Dictionary entries 57,60  
   dummy dictionary entries 55  
   processing in phase 10 42  
   processing in phase 20 52,53  
   processing in phase 21 60  
 file handling 29-34  
 file information block, VSAM (FIB) 68  
 file-name reference items 78  
 file-names  
   following RETURN 72  
   phase 30 processing 78,72  
   REF-text elements 144  
 File Section  
   Dictionary entries 58  
   phase 10 processing 43  
   phase 20 processing 53  
 FILE STATUS clause processing 41  
 FILEST routine 53  
 FILTBL table  
   description 299  
   optimizing VNs 114  
 FIL5BUF cell (COMMON) 281  
 FIPS (see federal information processing standard)  
 fixed routine, definition 471  
   (see also Report Writer Subprogram)  
 flag, FIPS 22  
 FLAGE 21  
 FLAGW 21  
 FLOW option  
   compiler overview 18-20  
   definition 21,126  
   diagram 477-480  
   file handling for 29-34  
   INIT3 instructions 438  
   parameters 35  
   phase 25 processing 69  
   phase 51 processing 105  
   phase 65 processing 126,127  
   PHZSW1 switch (COMMON) 284  
   procedure A-text processing 118  
   segmented program 122  
   TGTADTBL table 333  
   virtuals required 99  
 Flow Trace Table  
   allocation in phase 65 126,127  
   description 437  
 flowcharts 149-258  
   explanation of symbols 150  
   label directory 259  
   Report Writer Subprogram 240-258,453  
 FLOWSZ cell (COMMON)  
   description 279  
   tested in phase 65 126  
 FLUSH routine  
   description 47  
   relationship to other routines 45  
 FNTBL table  
   description 300  
   entries for USE sentences 50  
   finding file-names 50

- use for file section processing 42
- foldout diagrams 475
- forward branch tables
  - GNFWDBTB table
    - description 301
    - use in phase 62
  - PNFWDBTB table
    - description 313
    - use in phase 62 161
- free area 426
- FREE macro element 132
- FRS.GEN cell 447
- FRS.GRP cell 448
- FSTCDNM1 cell (COMMON) 282
- FSTEXT routine 53
- function-names for UPSI 58
  
- GANLNO cell 92
- GATXTV routine 99
- GCNTBL table 300
- GENASC routine 68
- GENDAT routine 75
- GENDTFCD routine 67
- GENDTFPR routine 68
- GENERATE statement
  - diagram of first RWS response 449
  - diagram of succeeding RWS responses 450
  - RWS response 449-451
- generated card number, REF-text element 144,145
- generated code for input/output verbs 454-469
- generated name (see procedure-names)
- GENERATED NAME CELLS field (PGT)
  - description 436
  - space allocation 117
  - space allocation with OPT 131
- GENNOD routine 92
- GENOP routine 78,72
- GENPAR routine 92
- GENTIM routine 92.1
- GET macro instruction 106
- GETALL routine 429
- GETCRD routine
  - function in phase 10 39
  - function in phase 11 50,48
- GETDLM routine
  - function in phase 10 39
  - function in phase 11 50
  - function in phase 12 45
- GETF2 routine 126
- GETNXT routine 92
- GETPTR routine 423
- GETWD routine
  - function in phase 10 39
  - function in phase 11 50,48
- GLORET routine 71
- GLOSRY routine
  - coding organization 71
  - description 71,72
- Glossary
  - DTF numbers 419
  - finding RWS elements 449
  - locating data 419
  - production 71
- GN number
  - allocation 117
  - with OPT 131
- data-name references 78
- DEBUG procedure 82
- definition 471
- deleting XSCRPT entries 95
- effect of ALTER/GO TO pair 82
- eliminating duplication 105
- file-name references 78
- from Declaratives Section 50
- generated for error declaratives 50
- generated for label declaratives 50
- locating RWS routines 449
- optimizing 114
- optimizing with OPT 134,129
- phase 50 processing 92
- phase 51 processing 105
- Q-routines 58
- segmented program 108
- simple IF statement 88
- GNATBL table
  - description 301
  - used by phase 62 130
- GNCALTBL table 301
- GNCTR cell (COMMON)
  - allocating GN numbers 117
  - assigning GN number 50
  - building GNTBL table 114
  - description 276
  - PERFORM statements 85
- GNFWDBTB table
  - description 301
  - use in phase 62 133
- GNLABTBL table
  - built by phase 62 134,133
  - description 302
- GNLBDTBL table
  - description 302
  - used by phase 64 134
- GNLIST work area 105
- GNSPRT routine
  - description 47,45
  - relationship to other routines 45
- GNTBL table
  - description 302
  - GN space allocation 117
  - optimizing GNS 114
- GO statement
  - effect on SEARCH verb 90
  - segmented programs 108
- GO TO NEXT SENTENCE statement
  - exit from SEARCH 90
  - generated by SEARCH verb analyser 90
- GO TO verbs 49
- GOBACK verb analyzer 105
- GP.nnnn cells 447
- GPLSTK table
  - description 303
  - generating Data A-text 58
  - use in Dictionary building 57
- group routine, definition 471
  - (see also Report Writer Subprogram)
- GRP.IND cell 447
- GSPICT routine 53
- GTEQ10K routine 127
- GTUNG cell (COMMON) 281
- GUI work area 53
- GVFNTBL table
  - description 304
  - phase 11 processing 50

GVMNTBL table  
 description 304  
 phase 11 processing 50

HASH table 73,421  
 hierarchy of arithmetic operators 87

I-O-CONTROL paragraph 41  
 ICTEXT work area 42  
 IDBRK routine 92  
 IDDCSN routine 39  
 IDENT routine 81  
 Identification Division  
 controlling processing 39  
 processing 39  
 processing overview 18  
 syntax analysis 43  
 translation, diagram 487  
 IDENTL cell (COMMON) 279  
 IDK field  
 base register assigning 122  
 data-name subscripts 94  
 defined 57  
 DOP work area 93  
 literal subscripts 94  
 optimizing subscripts 95  
 Working-Storage Section 57  
 IDLHN routine  
 DEBUG statement 82  
 PERFORM processing 85  
 P1-text processing 81  
 IDLH03 routine 92  
 ID1PTR work area 75  
 ID5 routine 92  
 IF routine 88  
 IF statements 88  
 ILBDABX0  
 linkage included by phase 51 106  
 ILBDDSP0  
 example of linkage codes 109  
 generating call 108  
 ILBDSEM0  
 generating call 108  
 ILBDSPM0 subroutine 108  
 ILBDVIO0 subroutine 110  
 ILBDVOC0 subroutine 110  
 incomplete Data A-text  
 completing elements 58  
 definition 471  
 generating in phase 20 53  
 incremented address element 139  
 INDEX cell (COMMON) 281  
 INDEX CELLS field (TGT)  
 contents 96  
 description 432  
 space allocation 113  
 index-names  
 direct indexing 96  
 indirect indexing 96  
 processing formula 96  
 processing in phase 50 96-98  
 indexed references, phase 50  
 processing 96-98  
 (see also index-names)

INDEX1 cell (COMMON)  
 description 279  
 TGT space allocation 113  
 INDKEY table  
 description 305  
 phases 22 and 30 processing 59  
 phase 30 searching 74  
 SEARCH Format-1 75  
 SEARCH Format-2 77  
 INDXTB table 305  
 IND2TBL table 306  
 initialization routines  
 concept 123  
 definition 471  
 INIT1  
 coding 430  
 description 431  
 functions 430  
 location in object module 430  
 purpose 123  
 INIT2  
 coding 436  
 description 437  
 functions 437  
 location in object module 430  
 purpose 123  
 INIT3  
 coding 438  
 description 437  
 functions 437  
 loading base registers 122  
 location in object module 430  
 purpose 123  
 RLD-TEXT 123  
 use of QGNTBL table 140  
 location 123  
 order written 123  
 INITIATE statement  
 locating INT-ROUT routine 452  
 phase 11 actions 44  
 response of RWS 440-445  
 INIT1  
 coding 430  
 description 431  
 functions 430  
 location in object module 430  
 purpose 123  
 INIT2  
 coding 436  
 description 437  
 functions 437  
 location in object module 430  
 purpose 123  
 INIT3  
 coding 438  
 description 437  
 functions 437  
 loading base registers 122  
 location in object module 430  
 purpose 123  
 RLD-TEXT 123  
 use of QGNTBL table 140  
 input  
 COMPUTE statements 87  
 error messages 418  
 phase 05 38.1  
 phase 06 38.2  
 phase 08 38.4

phase 12 46  
 phase 20 52  
 phase 22 56  
 phase 30 71  
 phase 50 92  
 phase 51 104  
 phase 70 146  
 phase 80 148  
 read from SYSIPT 14  
 SEARCH ALL statement 89  
 segmented program 121  
 input card number 55  
 input/output 28  
 input/output, control and, diagram 485  
 input/output error messages 418  
 Input-Output Section 40  
 input/output verbs  
   coding 454-469  
   influences on coding 106  
   phase 11 processing 49  
   phase 51 processing 106  
   register usage 106  
   VSAM 110  
 INSERT routine 428  
 INT-ROUT routine  
   description 441  
   flowchart 255  
 INT-routines, processing between phases 28  
 INTACC routine 422  
 Intermediate A-text, definition 471  
 Intermediate E-text  
   definition 471,104  
   input to phase 51 104  
 Intermediate Optimization A-text 104  
 Intermediate Procedure A-text 104  
 intermediate REF-text 107  
 intermediate results, maximum operand  
   size 97  
   (see also COMPUTE statement)  
 internal texts 341-384  
   A-text generation 99  
   generated by phase 05 38.1  
   generated by phase 12 46  
   generated by phase 20 52  
   generated by phase 22 56  
   identifier byte 341  
   list 16-18  
   phases producing 16-18  
   produced from Data Division 18  
   produced from Procedure Division 19  
   processing 14  
   use of work files 29-34  
 INTVIRT cell (COMMON) 282  
 INTVLC routine 58  
 INVALID KEY option 67  
 IOAREA area  
   address in DTF 68  
   buffer generation 68  
 IOPTBL table 306  
 IOPTR CELLS field (TGT)  
   description 433  
   space allocation 113  
 IOPTRCTR cell (COMMON)  
   description 279  
   TGT space allocation 113  
 IP-text 16  
 IPLT work area 53  
 IPRECS cell (COMMON) 286  
  
 IPT work area 53  
 IPTBUF area 410  
 ISTRUV routine 92.  
  
 KDECML field 40  
 KEYTAB table 306  
 KEYTBL table  
   description 307  
   processing SEARCH ALL 89  
 KILSUB routine 103  
  
 label records, effect on coding 106  
 LABELS cell (COMMON) 276  
 LABTBL table  
   description 307  
 LATACP routine 424  
 LATGRP routine 425  
 LATRNM routine 423  
 LATRPT routine 424  
 LCSECT cell (COMMON) 278  
 LD (see record description entries)  
 LD element  
   (see also record description entries)  
   Data IC-text entry 341  
   Dictionary entry 388  
 LDELNM routine 424  
 LDTEXT routine 53  
 LDTEXT work area 42  
 LENGTH OF SORTAB field (TGT) 433  
 LENGTH OF VN TBL field (TGT) 433  
 level-number entries  
   (see also record description entries)  
   Data IC-text entry 341  
   Dictionary entry 388  
 LIB option 22  
   effect on Lister 38.1  
   processing 37  
 library subroutines (see COBOL subroutines)  
 LIN.NUM cell 448  
 LIN.SAV cell 448  
 LINE-COUNTER cell 447  
 LINECNT cell contents 37  
   SYMDMP output 126  
 LINECT option 21,35  
 LINK option  
   definition 21  
   effect on buffers 36  
   phase 60 processing 111  
   phase 64 processing 141  
 LINKA routine  
   calling phase 01 24  
   functions 24,28  
   processing between phases 28,24  
   updating LINKCNT 24  
 linkage codes  
   definition 24  
   end of compilation 27  
   list 25  
   segmentation 34  
 linkage editor map  
   compiler 265-274  
   executed program example 420  
 Linkage Section  
   dictionary entries 58  
   processing in phase 10 43  
 LINKB routine

Procedure Block  
   (see also optimization (OPT) processing)  
   definition 472  
   for PERFORM statement 84-86  
 PROCEDURE BLOCK CELLS field (PGT)  
   description 436  
   space allocation 131  
 PROCEDURE BLOCK1 PTR field (TGT)  
   description 434  
 Procedure Division  
   encoding in P0-text 48  
   generated code location 436  
   listing 48  
   processing by phase 11 48  
   processing overview 477  
   sectional 50  
   translation, diagram 489  
 Procedure IC-text, definition 498  
   (see also P0-text; P1-text; P2-text)  
 PROCEDURE NAME CELLS field (PGT)  
   description 435  
   effect of ALTER 83  
   space allocation 117  
     with OPT 131  
 procedure-name reference 78  
 procedure-names  
   DEBUG attribute 82  
   definition 471  
   dictionary attributes 78  
   Dictionary entry 385  
   duplicate GNS 105  
   equate strings 105  
   GN number  
     data-name references 78  
     DEBUG procedure 82  
     definition 471  
     deleting XSCRPT entries 95  
     effect of ALTER/GO TO pair 82  
     eliminating duplication 105  
     file-name reference 78  
     from Declaratives Section 50  
     generated for error declaratives 50  
     generated for label declaratives 50  
     locating RWS routines 449  
     optimizing 114  
     optimizing with OPT 133,129  
     phase 50 processing 92  
     phase 51 processing 105  
     Q-routines 58  
     segmented program 108  
     simple IF statement 88  
     space allocation 117  
       with OPT 131  
   PERFORM statement 85  
   PN number  
     ALTER statements 82-84  
     DEBUG 82  
     definition 471  
     deleting XSCRPT entries 95  
     optimizing 114  
     phase 50 processing 92  
     phase 51 processing 105  
     segmented program 108  
     space allocation 117  
     without reference 105  
   REF-text elements 144,145  
   replacement of 78  
   segmented program 78  
     space allocation 117  
   space allocation with OPT 131  
   VN number  
     ALTER statements 82-84  
     deleting XSCRPT entries 95  
     definition 471  
     effect of ALTER/GO TO pair 82  
     GO string object 84  
     optimizing virtuals 116  
     PERFORM statement 84-86  
     phase 50 processing 92  
     phase 51 processing 105  
     segmented program 108,105  
     VNPTY table entry 114  
   processing between phases 24  
 PROCINDX table  
   built in phase 65 127  
   description 316  
 PROCTAB table  
   built in phase 65 126  
   description 404  
 PROC01 routine 45  
 PROC02 routine 47  
 PROGID cell (COMMON) 276  
 program breaks, use by phase 50 92  
 Program Global Table (PGT) 435-436  
   allocating space 116,130  
   base register 122  
   concept 435  
   definition 472  
   diagram of fields 435  
   fields  
     COUNT LINKAGE AREA 435  
     DEBUG LINKAGE AREA 435  
     DISPLAY LITERALS 436  
     GENERATED NAME CELLS 436  
     LITERALS 436  
     OVERFLOW CELLS 435  
     PROCEDURE BLOCK CELLS 436  
     PROCEDURE NAME CELLS 435  
     SUBDTF ADDRESS CELLS 436  
     VIRTUAL CELLS 435  
     VNI CELLS 436  
   GN allocation 117  
   listing 116  
   location in object module 435  
   LTLTBL table displacements 115  
   optimizing literals 115  
   optimizing space 112,116  
     with OPT 131  
   optimizing virtuals 116  
   overflow allocation 116  
   PN allocation 117  
   residence in segmented program 121  
   virtual allocation 116  
 PROGRAM-ID statement 39  
 program organization (flowcharts) 149-258  
 PROGSUM table  
   built in phase 65 127  
   description 396  
 PSHTBL table  
   description 317  
   IF statement 89  
 PSIGNT table  
   COMPUTE processing 87  
   description 317  
   IF processing 88

PSVCTR cell (COMMON)  
 description 280  
 TGT space allocation 113

PTRFLS table  
 description 317

PTYNO cell (COMMON) 278

PUT macro instruction 106

PUT routine 118

PUTDEF routine 105

P0-text  
 condition string 78-80  
 CORRESPONDING option 73  
 definition 472  
 description 351-358  
 encoded by phase 11 48  
 input to phase 30 71,72  
 Report Writer group routines 45  
 SEARCH Format-1 75-76  
 SEARCH Format-2 77  
 translation 72

P1-text  
 condition string 78-80  
 condition-name 78  
 CORRESPONDING option 73  
 data-name reference element 72  
 definition 472  
 description 359-363  
 determining type 78  
 dictionary attributes 78  
 file-name 78  
 generation 72  
 phase 40 processing 81  
 procedure-name 78  
 replacing names 78  
 SEARCH statement  
 SEARCH ALL 89  
 SEARCH Format-1 75-76  
 SEARCH Format-2 77  
 syntax analysis 91

P1BTBL table  
 description 318  
 Report Writer flag 44

P2-text  
 DEBUG 82  
 definition 472  
 description 364-371  
 DISPLAY coding 109  
 indexing 96  
 input to phase 51 104  
 MOVE statements 81  
 nested IF statement 89  
 ON statement 107  
 passed by phase 50 93  
 PERFORM output 86  
 phase 40 processing 81  
 presentation 93  
 SEARCH ALL statement 89  
 SEARCH statement 89  
 subscript optimizing 95  
 verb strings 81,92  
 VN for ALTER 82

Q-routine bit 78

Q-routines  
 (see also OCCURS...DEPENDING ON clause)  
 definition 498  
 discussion 436  
 EXAMINE verb 103  
 generation 58  
 identification elements 58  
 location in object module 430  
 location in segmented program 121  
 MOVE verb 103  
 object module 436  
 OD2TBL table, use of 42  
 optimization (OPT) processing 139  
 processing OCCURS...DEPENDING ON clauses 58  
 purpose 55  
 TRANSFER verb 103

QALTBL table, description 319

QBUILD routine 58

QFILE table  
 description 319  
 file-name references 78  
 phase 22 and 30 58  
 Q-routines, use in generating 58

QGNTBL table  
 built in phase 63 140  
 description 319

QITBL table  
 description 320  
 Q-routines, use in generating 58  
 phase 25 processing 69

QLTABL table, description 320

QNMTBL table  
 data-name from APPLY clause 41  
 description 320  
 File Section processing, use in 42  
 phase 10, use in 40,41

QRTN table  
 description 321  
 Q-routines, use in generating 58  
 used by phase 25 69

QSBLL table, description 321

QTBL table, description 321

QUALIF routine 75

qualified names 75

QUE routine 37

QUOTE option 21

QVAR table  
 description 322  
 finding GN numbers 78  
 generating Q-routines 58  
 phases 22 and 30 58

QVARBD routine 58

RCDTBL table  
 description 322  
 searching for REWRITE verbs 49  
 searching for WRITE verbs 49  
 use by phase 10 42

RD (report description entries) 55,57

RD statement  
 Data IC-text 343  
 Dictionary entry 385  
 phase 12 actions 45

RDFSTK table  
 description 323  
 use in dictionary building 57

RDF2 routine 126

RDSCAN routine 45

RDTXT routine 57

READ statement  
 error 72  
 output coding 460



phase 30 processing 72  
 phase 51 processing 106  
 READ verb strings 72  
 READFN routine 72  
 READF4 routine 55  
 READY verb 49  
 RECCONT routine 60  
 RECFORM parameter 61  
 RECFORM parameters supported 61  
 RECORD CONTAINS clause  
   determining record size 60  
   selecting record form 61  
 record description entries (LDs)  
   completed dictionary entries 57  
   dictionary entries 55  
   processing in phase 10 42  
   processing in phase 20 53  
   translation in phase 20 52  
 record form 60  
 record organization, effect on coding 105  
 record size 60  
 recording mode, effect on coding 105  
 RECORDING MODE clause, selecting record  
   form 61  
 REDEF routine 57  
 REDEFINES clause  
   entries in Dictionary 57  
   processing by phase 22 45  
 REF-text  
   contents 144,145  
   definition 472  
   phase 60 production 118  
   phase 61 processing 144,145  
 REFTBL table  
   description 421  
   overflow handling 144  
   phase 61 processing 144  
 REGATT area 75  
 register assignments  
   object module 122  
   optimizing 14 and 15 133  
   permanent 132  
   temporary 132  
 register handling routines 98  
 register save areas 469  
 register saving 418  
 register usage  
   compile time 411-417  
   execution time 419,420  
   saving 418  
 REGMTX table 118,122  
 RELADD cell (COMMON) 275  
 RELLOC cell (COMMON) 281  
 Relocation Dictionary (RLD)  
   phase 60 processing 123  
   virtuals 114  
 RENAMES clause  
   Dictionary entry adjustment 55  
   processing in phase 22 53  
 RENAMS routine 55  
 RENAMTB table  
   built by phase 22 59  
   description 323  
   used by phase 25 69  
 REPORT clause 44  
 report description entries (RDs) 55,57  
 Report Section  
   Dictionary entries 58  
   encountered by phase 10 41  
   listing 47  
   processing 44-47  
   related statements 45  
   translating 14  
 Report Section header, phase 10 actions 44  
 Report Writer feature (see phase 12; Report  
   Writer Subprogram)  
 Report Writer REDEFINES clause 45  
 Report Writer Subprogram (RWS) 440-453  
   COBOL word data-names 447-448  
   control-field save-area names 45  
   data-names 447-448  
   data items 447-449  
   definition 440  
   DET-ROUT routine, from GENERATE verb 44  
   E-Point data item, REDEFINES clause 45  
   elements 440-449  
   fixed routines 440  
   description 440  
   LST-ROUT  
     description 440  
     flowchart 256  
   WRT-ROUT  
     description 440  
     flowchart 246  
   1ST-ROUT  
     description 440  
     flowchart 241  
   flowcharts 240-258,453  
   general concept 392  
   group routines 446  
   CFF-ROUT  
     description 446  
     flowchart 257  
   CHF-ROUT 446  
   CTF-ROUT  
     description 446  
     flowchart 253  
   CTH-ROUT 446  
   description 446  
   DET-ROUT  
     description 447  
     flowchart 240  
     generating initial coding 44  
   PGF-ROUT  
     description 446  
     flowchart 244  
   PGH-ROUT  
     description 446  
     flowchart 251  
   RPF-ROUT  
     description 446  
     flowchart 258  
   RPH-ROUT  
     description 446  
     flowchart 247  
   INT-ROUT routine from INITIATE verb 44  
   LAST-ROLL instruction 440  
   location of data items 449  
   locations of elements 449-453  
   logic flow diagram 442-449  
   LST-ROUT routine from TERMINATE verb 44  
   nonstandard data-names 447-448  
   object module location 436  
   parametric routines 440-446  
   ALS-ROUT  
     description 441

flowchart 25  
 CTB-ROUT  
   description 441  
   flowchart 242  
 INT-ROUT  
   description 441  
   flowchart 255  
 RET-ROUT  
   description 441  
   flowchart 254  
 RLS-ROUT  
   description 446  
   flowchart 245  
 ROL-ROUT  
   description 441  
   flowchart 249  
 RST-ROUT  
   description 441  
   flowchart 248  
 SAV-ROUT  
   description 441  
   flowchart 252  
 USM-ROUT  
   description 441  
   flowchart 243  
 producing fixed routines 47  
 producing parametric routines 47  
 Report Writer verbs 448  
 residence in segmented program 121  
 response to verbs 448-452  
 routines 392-447  
 RPT.LIN data item, from REDEFINES  
   clause 45  
   1ST-ROUT routine, from GENERATE verb 44  
 REPORT-CALL verb 448  
 REPORT-ORIGIN verb 448  
   processing with OPT 139,140  
 REPORT-REORIGIN verb 448  
 REPORT-RETURN-n verbs 448  
 REPORT-SAVE-n verbs 448  
 REPTAB table 323  
 RERUN clause  
   effect on coding 106  
   processing in phase 10 41  
 RERUNN switch (COMMON) 283,41  
 RESERVE macro element 132  
 RET-ROUT routine  
   description 441  
   flowchart 254  
 RETURN verb 72  
 REWIND bit in DTFs 66  
 REWRITE statement  
   output coding 463  
   phase 11 processing 49  
   processing using table pointers 42  
 RGNCTR cell (COMMON)  
   description 278  
   used by phase 62 131  
 RLD-text 123,140  
 RLDTBL table  
   description 324  
   entries for Relocation Dictionary  
   (RLD) 116  
   phase 60 processing 123  
   phase 63 processing 139,140  
 RLS-ROUT routine  
   description 446  
   flowchart 245  
 RNMTBL table (phases 11, 12) 324  
 RNMTBL table (phase 22)  
   description 324  
   Dictionary building 57  
 ROL-ROUT routine  
   description 441  
   flowchart 249  
 ROLTBL table 325  
 root segment 48  
 ROUNDED clause  
   effect on EVAL string 88  
   temporary result 99  
 ROUTBL table 326  
 RPF-ROUT routine  
   description 446  
   flowchart 258  
 RPH-ROUT routine  
   description 446  
   flowchart 247  
 RPNCTR cell (COMMON)  
   description 279  
   used by phase 62 131  
 RPT.LIN cell 447  
 RPT.RCD cell 447  
 RPTSAV AREA field (TGT)  
   description 434  
   space allocation 113  
 RPTSAV cell (COMMON)  
   description 278  
   TGT space allocation 113  
 RST-ROUT routine  
   description 441  
   flowchart 248  
 RWRTBL table  
   checking for priming 45  
   description 326  
   priming by phase 10 44  
   use by phase 12 45  
 RWS (see Report Writer Subprogram)  
 RW1 counter 112  
 S.nnnn name cells 448  
 S-point name cells 448  
 SAME AREA clause  
   BUFTAB table entries 68  
   processing in phase 10 41  
 SAME clause 41  
 SAME RECORD AREA clause  
   BUFTAB table entries 68  
   effect on coding 106  
   processing in phase 10 41  
 SAME SORT AREA clause 41  
 SATBL table  
   description 327  
   processing in phase 10 41  
 SAV-ROUT routine  
   description 441  
   flowchart 252  
 SAVE AREA field (TGT) 434  
 SAVE AREA=2 field (TGT) 434  
 save areas 418  
 SA2CTR cell (COMMON) 278  
 SBL (secondary base location) 472  
 SBL CELLS field (TGT)  
   assigning SBL 57

description 434  
 OCCURS...DEPENDING ON clauses 58  
 Q-routines 57,59  
 space allocation 113  
 SBL number  
   assignment 57  
   definition 472  
   variably located fields 59  
 SBLCTR cell (COMMON) 277  
 scaling 118  
 SD (see sort description entries)  
 SD element  
   (see also sort description entries)  
   Data IC-text 341  
   Dictionary entry 385  
 SDSIZ cell (COMMON) 279  
 SDSRATBL table  
   description 327  
   use in phase 21 68  
 SDTF  
   address element 67  
   location 67  
   location in object module 430  
   location in storage 419  
   number assigned 67  
   order of addresses 419  
 SDTFADR, space in PGT 435  
   allocation 117,131  
 SDTFCTR cell (COMMON)  
   allocating PGT space 117  
   description 282  
 SDTXT routine 57  
 SEARCH ALL statement  
   example of processing 89-91  
   flow of execution 91  
   output 90  
   phase 40 processing 89-91  
 SEARCH routine  
   STSRCH routine processing 74  
   uniqueness of names 75  
 SEARCH statement  
   phase 30 processing 74-77  
   uniqueness of names 75  
 SEARCH verb analyzer 89  
 SEARCH verb, Format-1 75-76  
 SEARCH verb, Format-2 77  
 SEARCHALL routine 89  
 secondary base location (SBL),  
   definition 472  
 section-names, PERFORM delimiters 85  
 section 1. introduction 13-23  
 section 2. method of operation 24-148  
 section 3. program organization 149-258  
 section 4. directory 259-274  
 section 5. data areas 275-408  
 section 6. diagnostic aids 409-420  
 SEEK statement  
   output coding 467  
   phase 51 processing 106  
 SEGINDX table  
   built in phase 65 127  
   description 327  
 SEGLMT cell (COMMON)  
   checking in phase 51 108  
   description 279  
   setting contents 40  
   testing in phase 51 105  
   use in phase 11 48  
 SEGMENT element  
   (see also Debug-text)  
   phase 65 processing 126  
   use in building Debug File 126  
 SEGMENT-LIMIT clause  
   effect on root segment 121  
   priority number 40  
   processing in phase 11 48  
   storing priority number 40  
 segmentation control breaks 104  
 segmentation feature  
   ALTER statement 83  
   control break 78  
   definition 472  
   description 438-439  
   operations in phase 00 34  
   optimization  
     order of reading segments 140,141  
     procedure branching 108  
     root segment 141  
     segmentation control break 104  
   passing control breaks 92  
   PERFORM statement 85  
   phase 51 processing 104  
   priority checking 48  
   Procedure A-text processing 121  
   procedure branching 108  
   procédure-names 78  
   root segment 121  
   segment locations 438  
   segment names 39  
   SYMDMP processing 127  
   use of CONTBL and CONDIS tables 118  
 SEGNOTE function 104  
 SEGPROC routine 122  
 SEGSAVE cell 124  
 SEGTBL table  
   description 327  
   processing Procedure A-text 121  
   segment addresses 104  
   use by phase 60 34  
 SELECT clause 40  
 SELSCN routine 40  
 SEQ option 22  
 SEQERR cell (COMMON) 277  
 SET statement 96  
 SETL macro instruction 106  
 SETTBL table  
   description 328  
   processing verb strings 81  
 SEVTBL table 147  
 size of phases 13  
 size of records 60  
 SKPLNK routine  
   action at end of compilation 27  
   issuing EOJ macro 28  
 SMERGE routine 73  
 SMSTBL table 329  
 SNF routine 127  
 SNMTBL table 329  
 SORT CORE SIZE field (TGT) 434  
 sort description entries (SDs)  
   assignment of BL number 57  
   Dictionary entries 60  
   dummy Dictionary entries 55  
   processing in phase 10 42  
   processing in phase 20 53  
   processing in phase 21 60

SORT FILE SIZE field (TGT) 434  
 SORT MODE SIZE field (TGT) 434  
 SORT RET field (TGT) 434  
 SORT SAVE field (TGT) 434  
 SORT verb processing 149  
 SORT verb strings 73  
 SORTAB ADDRESS field (TGT) 434  
 SORTPROC routine 60  
 source element, definition 341  
 source program  
   branching verbs 47  
   commas as decimal points 40  
   debugging packet 82  
   Declaratives Section, USE  
   processing 106  
   file description entries (FDs)  
     assignment of BL number 55  
     Dictionary entries 57,60  
     dummy dictionary entries 55  
     processing in phase 10 42  
     processing in phase 20 53  
     processing in phase 21 60  
   file-name processing 72  
   FIPS flagging 148  
   group item processing 53  
   I/O verb processing 49  
   LD (see record description entries)  
   level-number entries (see record  
   description entries)  
   Lister processing 38.1-38.4  
   listing 39  
   LVL option 22  
   MOVE statement processing 81  
   procedure-names  
     point of definition 48  
     processing in phase 11 50  
   RD, partial dictionary entries 57  
   READ verb processing 72  
   record description entries (LDs)  
     completed dictionary entries 57  
     dictionary entries 57,55  
     processing in phase 10 42  
     processing in phase 20 55,57  
     translation in phase 20 52  
   reformatting 18  
   report description entries (RDs) partial  
   dictionary entries 57  
   scanning by phase 80 148  
   SDs (see sort description entries)  
   section, definition 121  
   SEGMENT, definition 121  
   sort description entries (SDs)  
     assignment of BL number 57  
     Dictionary entries 60  
     dummy Dictionary entries 55  
     processing in phase 10 42  
     processing in phase 20 52  
     processing in phase 21 60  
   verb processing in phase 11 49  
   verb profiles (VERBREF, VERBSUM) 23  
 SPACE option 22  
 SPACING cell (COMMON) 281  
 SPCREG area 75  
 special registers 75,78  
 SPECIAL-NAMES paragraph 40  
 SPNTBL table  
   ACCEPT entries 40,49  
   description 330  
   DISPLAY entries 40,49  
   entries by phase 10 40  
   REWRITE entries 49  
   WRITE entries 49  
   WRITE...AFTER entries 40  
 SRA routine 68  
 SRAMAX table  
   description 330  
   use in phase 21 68  
 SRATBL table (phase 10)  
   description 330  
   processing in phase 10 41  
   processing in phase 21 68  
 SRATBL table (phase 21) 331  
 SRCHKY table  
   description 331  
   syntax checking 59  
 SRCHTB routine 53  
 SRCTBL table 331  
 SRTTRERUN cell (COMMON) 286  
 SSATBL table  
   description 332  
   processing in phase 10 41  
 SSCRPT routine 96  
 START macro instruction 92  
 START statement  
   output coding 467  
   phase 51 processing 106  
 STATE option  
   CARDINDX table 290  
   compiler overview 20,475-480  
   Debug File 395-405  
   Debug-text 384  
   definition 22,126  
   file handling 29-34  
   parameters 35  
   phase 50 processing 99  
   phase 51 processing 105  
   phase 65 processing 126,127  
   PHZSW2 switch (COMMON) 284  
   Procedure A-text processing 118  
   segmented program 122  
   SEGINDX table 327  
   SWITCH field of TGT 434  
   TGT fields 112  
   TGTADTBL table 333  
   verb processing for 93  
   verb strings 105-107  
   virtuals required 99  
 STATIC routine 428  
 STOP RUN verb analyzer 105  
 storage  
   allocation for PGT 112  
   allocation for PGT with OPT 130,131  
   allocation for TGT 112  
   allocation for TGT with OPT 130  
   COBOL space 426  
   contents during phase 20 52  
   contents during phase 22 55  
   data locations 418  
   Dictionary area 425  
   Dictionary location 428  
   DTF location 418  
   free area 428  
   layout during compilation  
     buffer contents 410  
     compiler 13  
     current phase 409

location of information in 13  
 required for phases 13  
 STRING table  
 COMPUTE processing 89  
 description 332  
 MOVE statements 81  
 processing verb strings 81  
 purpose 81  
 STSRCH routine 72,74  
 STXIT option  
 definition 22  
 processing OPEN statements 106  
 SUBADR CELLS field (TGT)  
 assigning subscript save cells 98  
 description 434  
 ON verb processing 105  
 space allocation 113  
 SUBCTR cell (COMMON)  
 description 281  
 ON verb processing 105  
 TGT space allocation 113  
 SUBDTF ADDRESS CELLS field (PGT)  
 description 436  
 locating on SDTF 419  
 space allocation 117  
 subject hierarchy, definition 472  
 subroutine (see COBOL subroutines)  
 SUBSCRIPT string 81,82  
 SUBSCRIPT verb 93  
 subscripted MOVE, output from phase 40 82  
 subscripted references  
 calculating addresses 93-95  
 data-names 94  
 formula 94  
 literals 94  
 mixed literals and data-names 95  
 rules 93  
 use of PICTURE 93  
 DATATAB entries 402  
 optimizing 95  
 resolving in phase 50 93-95  
 SUBTRACT verb processing 72  
 SUMTBL table 332  
 SUPMAP  
 definition 22  
 determining error severity 111  
 effect on PGT listing 116  
 effect on phase 60 118  
 effect on phase 70 146  
 listing TGT 112  
 literal pool 115  
 SWITCH cell (COMMON) 283  
 set by phase 65 112  
 SWITCH field (TGT) 434  
 SWITCH1X cell (COMMON) 285  
 SWITV2 cell (COMMON) 282  
 SXREF option  
 definition 22  
 effect on file SYS004 123  
 output of phase 60 11  
 phase 60 processing 111,118  
 phase 61 processing 144,145  
 phase 64 processing 141  
 SYM option  
 definition 21,71  
 effect on PGT listing 116  
 SYMDMP option  
 buffer assignments 36  
 CARDINDX table 290  
 compiler overview 20,475-480  
 Debug File 395-405  
 Debug-text 384  
 definition 22,126  
 Dictionary 19  
 E-text element 382  
 file handling for 29-34  
 INIT3 instructions 438  
 OBODOTAB processing 69  
 OD2TBL table 42  
 parameters 35  
 phase 25 processing 69  
 phase 50 processing 99  
 phase 51 processing 105  
 phase 60 processing 118,122  
 phase 65 processing 126,127  
 PHZSW2 switch (COMMON) 284  
 Procedure A-text processing 118  
 segmented program 122  
 Q-routine generation 58  
 SEGINDX table 327  
 SWITCH cell of COMMON 283  
 SWITCH field of TGT 434  
 tables built by phase 22 59  
 TGT fields 112  
 TGTADTBL table (TIB18) 333  
 verb processing for 93  
 verb strings 105,106  
 VARLTBL table 336  
 virtuals required 99  
 syntax analysis in phase 10 43  
 syntax errors in phase 05 38.1  
 SYNTAX option  
 (see also CSYNTAX option)  
 definition 23  
 phase 01 processing 35  
 SYSIPT  
 buffer control block 410  
 buffer sizes 35,37  
 opening 37  
 SYSLNK  
 buffer control block 462  
 buffer sizes 35,37  
 opening 37  
 SYSLST  
 buffer sizes 35,37  
 change to SYS006 38  
 control block 410  
 cross-reference list 144,145  
 E-text 146  
 glossary listing 71  
 listing by phase 07 38.3  
 listing by phase 08 38.4  
 listing by phase 10 39  
 listing of Identification Division 40  
 listing of Report Section 44  
 literal pool 115  
 opening 37  
 output of options 21  
 PGT listing 116  
 phase 70 146  
 phase 80 148  
 Procedure A-text output 118  
 SYSnmm files  
 buffer sizes 35,37  
 opening 37

SYSPCH  
   buffer control block 410  
   buffer sizes 35,37  
   opening 37  
 SYS004 contents 123  
 SYS005 file  
   (see also Debug File)  
   buffer assignments 36  
   compiler file handling 29-34  
   phase 25 processing 69  
   phase 65 processing 127,128  
   sequence of phases 28  
 SYS006 file 38,21  
  
 TA LENGTH field (TGT)  
   description 472  
 table  
   (see also compiler tables)  
   definition 472  
   maximum size 426  
 table and dictionary area 421  
 table and dictionary handling 421-429  
 table area 427  
 Table Area Management Executive Routines  
   (see TAMER)  
 Table Area Management Map (see TAMM)  
 table control fields 425-426  
 table formats 287-340  
 table handling (see TAMER)  
 Table Information Block (see TIB)  
 table of contents diagram 481  
 tables used by phases 263,459  
 TABREL routine 428  
 TALLY field (TGT) 434  
 TAMAREA cell 37  
 TAMEIN routine 422,426  
 TAMEOP routine 428  
 TAMER  
   control fields 425-426  
   (see also TIB; TAMM; MASTAM)  
   definition 472,421  
   error handling 34  
   function 34  
   GETALL routine 144  
   location 425,421  
   processing between phases 24-26  
   routines  
     DICSPC 427  
     GETALL 429  
     INSERT 428  
     MOVDIC 427  
     PRIME 427  
     STATIC 428  
     TABREL 428  
     TAMEIN 426,422  
     TAMEOP 428  
     TBGETSPC 427  
     TBREADIC 429  
     TBSPIII 428  
     TBWRITE 428  
   saving registers 418  
 TAMER space, definition 472  
 TAMER tables 287-340  
 TAMM  
   address 425  
   contents 425  
   description 425  
   format 425  
   locating tables 410  
   location in storage 426  
 TAMNAD cell (COMMON) 275  
 Task Global Table (TGT) 430,432-435  
   base register 122  
   concept 430  
   diagram of fields 432  
   fields 432-435  
     A(INIT1) 432  
     BL CELLS 432  
     BLL CELLS 432  
     CHECKPT CTR 432  
     COUNT CHAIN ADDRESS 432  
     COUNT TABLE ADDRESS 432  
     CURRENT PRIORITY 432  
     DEBUG TABLE PTR 432  
     DEBUG TABLE 432  
     DTFADR CELLS 432  
     ENTRY-SAVE 432  
     INDEX CELLS 432  
     IOPTR CELLS 433  
     LENGTH OF SORTTAB 433  
     LENGTH OF VN TBL 433  
     NSTD-REELS 433  
     ONCTL CELLS 433  
     OVERFLOW CELLS 433  
     PARAM CELLS 433  
     PFMCTL CELLS 433  
     PFMSAV CELLS 433  
     PGM ID 433  
     PGT-VN TBL 433  
     PROCEDURE BLOCK1PTR 434  
     RPTSAV AREA 434  
     SAVE AREA 434  
     SAVE AREA=2 434  
     SBL CELLS 434  
     SORT CORE SIZE 434  
     SORT FILE SIZE 434  
     SORT MODE SIZE 434  
     SORT RET 434  
     SORT SAVE 434  
     SORTAB ADDRESS 434  
     SUBADR CELLS 434  
     SWITCH 434  
     TA LENGTH 434  
     TALLY 434  
     TEMP STORAGE 434  
     TEMP STORAGE-2 435  
     TEMP STORAGE-3 435  
     TEMP STORAGE-4 435  
     TGT-VN TBL 435  
     UPSI SWITCHES 435  
     VLC CELLS 435  
     VN CELLS 435  
     WORKING CELLS 435  
     XSA CELLS 435  
     XSASW CELLS 435  
   location of fields 112  
   location in object module 430  
   overall length 112  
   residence in segmented program 121  
   storage allocation 112  
   UPSI bit switches 58  
   using PN, GN, and VN 58-59  
   variable-length fields  
     COMMON counters 112  
     determining location 112  
     discussion 112

lengths 58  
 TBGETSPC routine 427  
 TBREADIC routine 429  
 TBSPILL routine 428  
 TBWRITE routine 428  
 TEMP STORAGE field (TGT)  
   description 434  
   space allocation 113  
 TEMP STORAGE-2 field (TGT)  
   description 435  
   space allocation 113  
 TEMP STORAGE-3 field (TGT)  
   description 435  
   space allocation 113  
 TEMP STORAGE-4 field (TGT)  
   description 435  
   space allocation 113  
 temporary register assignments 132  
 temporary result 99  
 temporary storage 98  
 TER.COD cell 309  
 TERMINATE statement  
   diagram of RWS response 452  
   phase 11 actions 44  
   RWS response 449  
 terminating compilation 24  
 text formats 341-384  
 texts (see internal texts)  
 TGT (see Task Global Table)  
 TGT-VN TBL field (TGT) 435  
 TGTADTBL table 333  
 TGTINT routine 112  
 THRU option 85  
 TIB (Table Information Block)  
   address 425  
   contents 425  
   definition 472  
   description 425  
   format 425  
   locating tables 410  
 TIB cells (COMMON) 275  
 TIME cell (COMMON) 282  
 TMCNTBSZ cell (COMMON) 282  
 transient area, definition 472  
 transient area for segmented program 438.1  
 TRMNATE routine 27  
 TRUNC option 23  
 TS area 98  
 TS cells 98  
 TSMAX cell (COMMON)  
   description 276  
   relation to XAVAL table 98  
   TGT space allocation 113  
 TSTWRO routine 71  
 TS2MAX cell (COMMON)  
   description 277  
   TGT space allocation 113  
 TS3MAX cell (COMMON)  
   description 279  
   TGT space allocation 113  
 TS4MAX cell (COMMON)  
   description 279  
   TGT space allocation 113  
 TXPNCH routine 127  
 TYPE IS DETAIL clause, locating DET-ROUT  
   routine 452  
 unrecoverable error conditions 34  
 UPSI  
   bit switches 58  
   definition 473  
 UPSI routine 58  
 UPSI SWITCHES field (TGT)  
   contents 58  
   description 435  
 UPSI-n clause 40  
 UPSTBL table  
   description 334  
   entries by phase 10 40  
   use in Dictionary building 58  
 USAGE clause 53  
 usage for group items 53  
 USE statement  
   ON clauses 50  
   output coding 469  
   phase 11 actions 44  
   phase 51 processing 106  
 USM-ROUT routine  
   description 441  
   flowchart 243  
 USNGTBL table 335  
  
 VALGEN routine 53  
 VALGRP table  
   description 335  
   generating Data A-text 58  
   produced and passed by phase 20 52  
 VALTRU table  
   description 335  
   produced and passed by phase 20 52  
   use in phase 22 59  
 VALUE clause  
   Data A-text 58  
   data items initialized 430  
   phase 20 processing 53  
 VALUE IS SERIES clause  
   syntax checking 59  
   VALGRP and VALTRU table entries 52  
 VARLTBL table  
   built by phase 22 59  
   description 336  
   used by phase 25 69  
 VARYTB table 336  
 VCON 67  
 verb analyzer 91  
 verb analyzer routine 81  
 verb blocks, defining 92  
 verb code list 353-354  
 VERB option 23  
 verb processor 93  
 verb strings  
   (see also input/output verbs;  
   nonarithmetic verb strings)  
 ALTER statement 82-83  
 COMPUTE statement 87-88  
 continuation strings 81  
 definition 473  
 distinguishing numerics 93  
 EQUATE 85  
 GO 84  
 IF statements 88  
 MOVE statement 81  
 number of operands 81

PERFORM statement 84-87  
   phase 40 processing 81  
   phase 50-phase 51 103  
   phase 50 processing 92-97  
   p2-text 92  
 SEARCH ALL statement 89-91  
 VERBDEF Tamer table  
   description 336.1  
   phase 11 processing 48  
   phase 22 processing 58  
 VERBREF option 23,144  
 VERBSUM option 23,144  
 VIRCTR cell (COMMON)  
   description 276  
   displacement of virtuals 116  
   virtual number assigning 99  
 VIRPTR table  
   after VIRTUAL allocation 117  
   description 337  
   optimizing virtuals 116  
   virtual allocation 116  
 VIRRTN routine 116  
 VIRTUAL CELLS field (PGT)  
   description 435  
   space allocation 116  
   space allocation with OPT 131  
 virtual references optimizing 116  
 virtuals  
   optimizing for PGT 116  
   phase 50 processing 99  
 VLC CELLS field (TGT)  
   description 435  
   space allocation 113  
 VLCTTR cell (COMMON)  
   description 277  
   OCCURS...DEPENDING ON clauses 58  
   TGT space allocation 113  
 VN CELLS field (TGT)  
   description 435  
   effect of ALTER 83  
   space allocation 113  
 VN number  
   ALTER statements 82  
   deleting XSCRPT entries 95  
   definition 471  
   effect of ALTER/GO TO pair 82  
   GO string object 84  
   optimizing virtuals 116  
   PERFORM statement 84  
   phase 50 processing 92  
   phase 51 processing 105  
   segmented program 108,104  
   VNPTY table entry 114  
 VNCTR cell (COMMON)  
   description 280  
   PGT space allocation, 117  
     with OPT 131  
   VN number for PERFORMs 85  
 VNI CELLS field (PGT)  
   description 436  
   space allocation 117  
     with OPT 131  
 VNILOC cell (COMMON)  
   description 281  
   PGT space allocation 117  
     with OPT 131  
 VNLOC cell (COMMON)  
   description 281  
   TGT space allocation 113  
 VNPNTBL table  
   built by phase 62 130  
   description 337  
 VNPTY table  
   description 338  
   table building 130,114  
 VNTBL table  
   ALTER statements 82  
   ALTER/GO TO pair 82  
   description 338  
   keeping track of VNs 85  
   PERFORM processing 84  
   processing verb strings 81  
 VRBCNT cell (COMMON) 282  
 VSAM file information block (FIB) 68  
 VSAM file processing  
   phase 10 processing 40  
   phase 11 processing 50  
   phase 51 processing 110  
 VSAM input/output verbs 110  
  
 WCMAX cell (COMMON) 276  
 WHEN clause effect on SEARCH verb 90  
 WKAREA work area 75  
 work files 14,16-18  
 WORKING CELLS field (TGT)  
   description 435  
 Working-Storage Section  
   address elements 58  
   assignment of BL number 57  
   data items location 430  
   processing in phase 10 43  
 WRITE statement  
   output coding 463  
   phase 11 processing 49  
   phase 51 processing 106  
   processing using table pointers 42  
 WRITE...ADVANCING statement  
   entries in SPNTBL table 40  
 WRITE-ONLY files 71  
 WRITE-ONLY switch 71  
 WRSYS4 routine 92.1  
 WRT-ROUT routine  
   description 440  
   flowchart 246  
 WSDEF cell (COMMON) 278  
  
 X code  
   definition 24  
   end of compilation 28  
   list 25  
   segmentation 34  
 XAVAL table  
   description 338  
   temporary storage assignment 98  
 XFREER routine 98  
 XINTR table  
   arithmetic verb translating 97  
   description 339  
 XNORML routine 147  
 XPUT routine 147



XREF option  
  definition 21  
  effect on SYS004 123  
  phase 60 processing 111,118  
  phase 61 processing 144,145  
  phase 64 processing 141

XREF-text  
  data-name DEF elements 58  
  description 383  
  location on file SYS004 123  
  output of phase 60 111  
  processing in phase 60 123  
  processing in phase 61 144,145  
  use in cross-reference listing 144,145

XREGNO cell 98

XSA CELLS field (TGT)  
  description 435  
  space allocation 113

XSACTR cell (COMMON)  
  description 280  
  TGT space allocation 113

XSASW CELLS field (TGT)  
  description 435  
  space allocation 113  
  ON verb processing 107

XSCOMP routine  
  description 94  
  indexing 96

XSCRPT table  
  data-name subscripts 94  
  deleting entries 94,103  
  description 339

  direct indexing 96  
  indirect indexing 96  
  literal subscripts 94  
  optimizing subscripts 95

XSPRO routine 103

XSSNT table  
  description 340  
  direct indexing 96  
  optimizing subscripts 95

XSWCTR cell (COMMON)  
  description 280  
  ON verb processing 107

XU6REC work area 147

Y code  
  definition 24  
  end of compilation 28  
  list 25  
  segmentation 34

ZWB option 23

01-level Report Writer statements 45

1ST-ROUT routine  
  description 440  
  flowchart 241

IBM DOS/VS COBOL  
Compiler Program Logic  
LY28-6423-1

**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, job title, and business 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 439  
Palo Alto, California

---

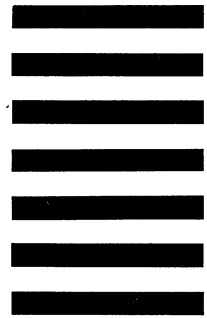
**Business Reply Mail**

No postage necessary if mailed in the U.S.A.

---

Postage will be paid by:

**IBM Corporation  
General Products Division  
Programming Publishing—Department J57  
1501 California Avenue  
Palo Alto, California 94304**



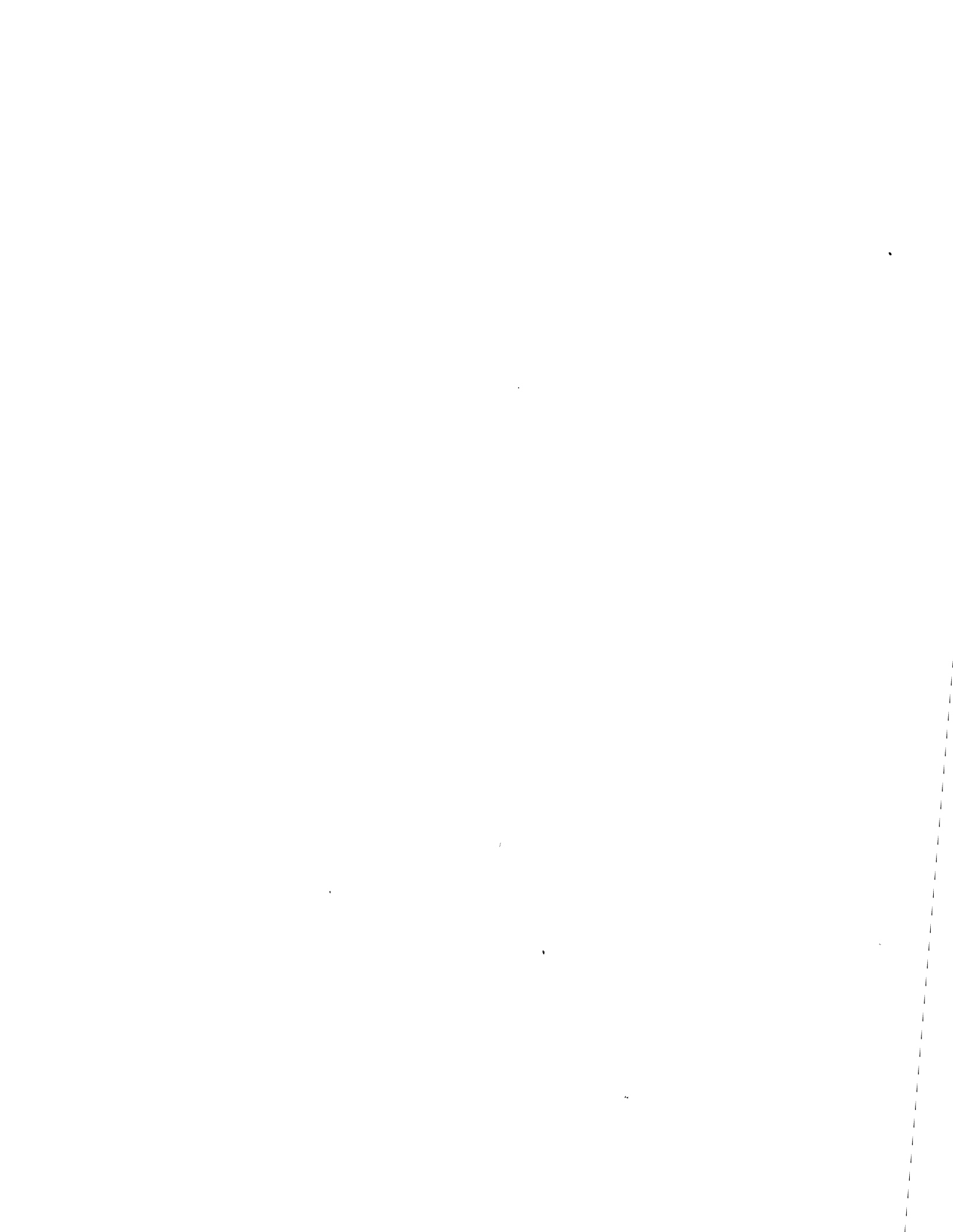
Fold and Staple

IBM DOS/VS COBOL Comp. PLM Printed in U.S.A. LY28-6423-1



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)





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)