

Program Logic

Version 8.1

IBM System/360 Time Sharing System Assembler

This publication describes the internal logic of the IBM System/360 Time Sharing System (TSS/360) Assembler Program (also referred to as "the assembler"). The assembler processes a group of statements written according to the rules of the TSS/360 Assembler Language into a TSS/360 program module. A general explanation of the four phases of language processing in the assembler is provided, followed by individual routine descriptions and flowcharts.

A general understanding of TSS/360 and the rules of the TSS/360 Assembler Language is assumed. Prerequisite to and co-references with this publication are: IBM System/360 Time Sharing System: Concepts and Facilities, and IBM System/360 Time Sharing System: Assembler Language.

This publication is intended for use by system programmers involved in changing system code and in altering the assembler design.

Third Edition (September 1971)

This is a major revision of, and makes obsolete, GY28-2021-1 and Technical Newsletters Y28-3100, GN28-3129, and GN28-3138. There are numerous technical changes to this publication, both in the flowcharts and routine descriptions. The major changes are summarized below:

- The CXD (CEVCX) routine has been added to Phase I to scan for the presence of CXD instructions. A new address constant, Q, has been introduced, and a DXD item has been added to the main dictionary. Several routines have been altered to process the new CXD and DXD instructions.
- The EQU instruction now permits length and type attribute operands. The EQU (CEVQU) and EQUATE (CEVEQ) routines have been altered to process the new operands.
- The USE/DROP (CEVUD) routine has been changed to process a null operand on a DROP instruction. If this situation occurs, all previously designated base registers are dropped.
- The EBCDIME (CEVET) routine is obsolete and has been deleted.

This edition is current with Version 8 Modification 1 of the IBM System/360 Time Sharing System (TSS/360), and remains in effect for all subsequent versions or modifications of TSS/360 unless otherwise noted. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters. Before using this publication, refer to the latest edition of IBM System/360 Time Sharing System: Addendum, GC28-2043, which may contain information pertinent to the topics covered in this edition. The Addendum also lists the editions of all TSS/360 publications that are applicable and current.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

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 reader's comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Time Sharing System/360 Programming Publications, Department 643, Neighborhood Rd., Kingston, N.Y. 12401

The first section of this program logic manual is an introductory discussion of the overall concepts of the Time Sharing System/360 (TSS/360) assembler program. A number of sections, each associated with a major component of the program, follows the introduction.

In addition to a general summary of the assembler's functions, the introduction describes the external interfaces between the assembler and

- Language Processor Control (LPC)
- Symbolic library service routines
- VISAM management service routines

Through its virtual memory management routines, which issue GETMAIN and FREEMAIN macro instructions, the assembler also interfaces with the Virtual Storage Allocation service routine (CZCGA).

Section 1 describes the overall flow of control only from the language processor control (LPC) to the phase control level.

Sections 2 and 3 contain summary level material; Section 2 contains summary by phases, and Section 3 by instruction type.

Sections 4 through 10 describe the relationships between phases and routines. The routine relationships of each major component (phase) are shown in an illustration following the introduction to that phase. All routines are represented by a box and an entry in the decision table supporting the illustration. All relationships between routines are shown with arrows. Except in the introduction and in Section 1, the arrow represents a call and a return to the routine from which the arrow points. In the introduction and in Section 1, the arrow represents the flow of control. In Section 5, the arrow may represent the recursive entry of one routine into another. This exception is noted in the introduction to Section 5.

The detailed flowcharts for the routines are presented in Section 11, arranged in the same order as the routine descriptions.

Section 12 describes all the tables, table entries, and listing formats referred to in this manual.

Section 13 describes the assembler's virtual memory management routines. These routines manage virtual storage requisition and return, issuing GETMAIN and FREEMAIN macro instructions when necessary.

The routine relationships are shown in terms of levels. A called routine is considered to be one level lower than the calling routine. Every box in each routine relationship's illustration has an Arabic numeral in the right-hand corner, indicating the lowest level at which the module may be called. Phase control routines are considered to be level 1.

The illustrations showing the routine relationships are supported by decision tables. Each routine in an illustration is supported by an entry in the corresponding decision table, which lists the conditions under which that routine calls other routines. The decision table entries are placed in their level order; within each level, the entries are arranged alphabetically by mnemonic name.

Upon completion of this manual, the reader will have a comprehensive knowledge of the internal functions of the TSS/360 assembler program. If more detailed knowledge is required, the program listings should be consulted.

PREREQUISITE PUBLICATIONS

Effective use of this manual requires knowledge of the information contained in the following manuals:

IBM System/360 Time Sharing System:
Concepts and Facilities, GC28-2003

IBM System/360 Time Sharing System:
Assembler Language, GC28-2000

In addition, the following publications may be consulted:

IBM System/360 Time Sharing System:
Dynamic Loader PLM, GY28-2031

IBM System/360 Time Sharing System:
Program Control System PLM, GY28-2014

CONTENTS

SECTION 1: INTRODUCTION	1
Purpose of the TSS/360 Assembler Program	1
System Environment	1
Organization and Overall Function of the Assembler	2
Syntax Analysis	3
Macro Instruction Processing	3
Assignment of Location Counter Values	4
Program Reordering	4
Machine Instruction Synthesis	5
Post-Processing	5
Assembler Functions	5
Assembler Control Routine (Interface with LPC)	5
User Virtual Storage Required by Assembler	7
Working Storage Areas	7
Work Area 1	7
Work Area 2	7
Work Area 3	8
PMD Text, ISD and External Names List Storage Areas	8
Characteristics of Assembler Routines	8
SECTION 2: ASSEMBLER FUNCTIONAL DESCRIPTION	9
Phase I Functional Description	9
Phase IIA Functional Description	13
Phase IIB Functional Description	16
Phase IIC Functional Description	20
Phase III Functional Description	22
Phase IV Functional Description	26
SECTION 3: ASSEMBLER FUNCTION BY INSTRUCTION TYPE	28
Introduction	28
Machine Instructions	28
Macro Instructions	28
Assembler Instructions	28
SECTION 4: ASSEMBLER MASTER CONTROL	32
Introduction	32
AC -- Assembler Control (CEVAC)	33
SECTION 5: PHASE I	36
Introduction	36
Routines	36
PHASE I -- Phase I Control (CEVPA)	36
STAN -- Statement Analyzer (CEVST)	36
REED -- Obtain Next Source Statement (CEVRD)	53
GETOP -- Collect and Identify Operation Code (CEVGP)	54
SUBOP -- Substitute into Operation Code Field (CEVSP)	55
CATOP -- String Substitution Control (CEVCP)	55
MIP -- Machine Instruction Operand Scan (CEVMP)	56
BASCAN -- Basic Source Language Scan (CEVBS)	56
AGO/AIF -- AGO/AIF Instruction Scan (CEVGO)	57
ANOP -- ANOP Instruction Scan (CEVAN)	58
CCW -- CCW Instruction Scan (CEVCW)	58
CNOP -- CNOP Instruction Scan (CEVCN)	59
CXD -- CXD Instruction Scan (CEVCX)	59
SECT -- Control Section Instruction Scan (CEVCT)	59
COPY -- COPY Instruction Processor (CEVCY)	60
DC/DS -- DC/DS Instruction Scan (CEVDC)	61
EJECT -- EJECT Instruction Scan (CEVEJ)	61
END -- END Instruction Scan (CEVND)	61
ENTRY -- ENTRY Instruction Scan (CEVEY)	62
EQU -- EQU Instruction Scan (CEVQU)	62
EXTRN -- EXTRN Instruction Operand Scan (CEVXN)	62

GBLx/LCLx -- Global/Local Symbol Instruction Scan (CEVGL)	63
ICTL -- ICTL Instruction Scan (CEVIC)	64
ISEQ -- ISEQ Instruction Scan (CEVIQ)	64
LTORG -- LTOrg Instruction Scan (CEVLG)	65
MACRO -- MACRO Instruction Scan (CEVMC)	65
MEND/MEXIT -- MEND/MEXIT Instruction Scan (CEVMX)	65
MNOTE -- MNOTE Instruction Scan (CEVMN)	66
ORG -- ORG Instruction Scan (CEVRG)	66
PRINT -- PRINT Instruction Operand Scan (CEVPR)	67
PUNCH -- PUNCH Instruction Scan (CEVPH)	67
REPRO -- REPRO Instruction Scan (CEVRE)	67
SETX -- SET Instruction Scan (CEVSE)	68
SPACE -- SPACE Instruction Scan (CEVCE)	68
TITLE -- TITLE Instruction Scan (CEVTI)	69
USE/DROP -- USING and DROP Instructions Scan (CEVUD)	69
MACREF -- Macro Reference Processor (CEVRF)	70
MACDEF -- Macro Definition Processor (CEVDF)	70
CSCAN -- Constant Scan (CEVCS)	71
SSCAN -- String Substitution Scan (CEVSS)	72
EVAL -- Expression Evaluator (CEVEV)	72
PSCAN -- Parameter Item Analyzer (CEVPS)	77
EBIN -- Binary Self-Defining Term Generator (CEVGB)	78
EDEC -- Decimal Self-Defining Term Generator (CEVGD)	78
EHEX -- Hexadecimal Self-Defining Term Generator (CEVGH)	78
ECHAR -- Character Self-Defining Term Generator (CEVGC)	79
SLIT -- Scan for Literal Operand (CEVSL)	79
DLPM -- Dictionary Lookup and Put (CEVLP)	79
DEFSYM -- Define Location Symbol (CEVSY)	80
DIAG -- Diagnostic Message Processor (CEVDX)	80
DLKT -- Lookup Temporary Dictionary Item (CEVTK)	81
DPUT -- Put Item in Temporary Dictionary (CEVTP)	81
MACLKT -- Macro Name Dictionary Lookup (CEVLM)	82
MACPUT -- Macro Name Dictionary Put (CEVTM)	82
DLKM -- Main Dictionary Lookup (CEVKM)	82
SECTION 6: PHASE IIA 83	
Introduction	83
Conversational Control	83
Routines	83
PHASE IIA -- Phase IIA Control (CEVPB)	83
PARAMAC -- Macro Parameter Processor (CEVPM)	86
SECTION 7: PHASE IIB 88	
Introduction	88
Routines	88
PHASE IIB -- Phase IIB Control (CEVPC)	88
LOCATE -- Location Counter Assignment (CEVLC)	91
ORIGIN -- Location Counter Reset (CEVGN)	92
POOLIT -- Literal Pooling Processor (CEVPL)	92
EQUATE -- Assign Value to Name (CEVEQ)	93
RESCON -- Resolve Conditional Alignment (CEVRS)	93
RESLIT -- Literal Resolution Processor (CEVRL)	94
SECTION 8: PHASE IIC 95	
Introduction	95
Routines	96
PHASE IIC -- Phase IIC Control (CEVPD)	96
USET -- USING Table Processor (CEVUP)	97
DRSET -- DROP Table Processor (CEVDR)	97
SECTION 9: PHASE III 99	
Introduction	99
Routines	99
PHASE III -- Phase III Control (CEVPE)	99
SLS -- Source Listing Processor (CEVSX)	104
GATEW -- Interface with VISAM PUT or GTWRC Macro (CEVGW)	104
ENDPR -- Module Entry Point Processor (CEVEP)	104
MOPR -- Phase III Machine Operation Processor (CEVMO)	104

GETVAL -- Obtain Relocatable Value (CEGV)	.105
USEVAL -- Compute Using Register (CEVUV)	.106
LIST -- Object Program Listing (CEVLS)	.106
CCWTXT -- Phase III CCW Instruction Processor (CEVCC)	.107
PUTVAL -- Relocatable Output Value Processor (CEVPV)	.107
DCTXT -- Phase III Constant Processor (CEVDP)	.108
ADCON -- Address Constant Processor (CEVAD)	.109
LITXT -- Phase III Literal Pooling Processor (CEVLT)	.109
CSDPR -- CSD Processor (CEVCD)	.110
SECTION 10: PHASE IV	.112
Introduction	.112
Routines	.112
PHASE IV -- Phase IV Control (CEVPF)	.112
XREF -- Cross-Reference Listing Processor (CEVXF)	.112
STED -- Symbol Table Editor (CEVSR)	.114
ISDPR -- ISD Processor (CEVSD)	.114
PMDLS -- Program Module Dictionary Listing Processor (CEVMD)	.115
ISDSA -- ISD List Processor (CEVSA)	.115
SECTION 11: FLOWCHARTS	.116
SECTION 12: TABLES, TABLE ENTRIES, LISTING FORMATS	.260
Main Dictionary	.260
Basic Format	.260
Absolute Value Item	.261
Relocatable Value Item	.261
DXD Item	.262
Complex Value Item	.263
External Name Item	.264
Control Section Item	.265
Entry Trailer Item	.266
Literal Item	.266
Transitive Item	.268
Local Variable Symbol Items	.268
Global Variable Symbol Items	.269
Sequence Symbol Item	.272
Logical Order File (LOF)	.272
Machine Operation Entry	.272
Macro Instruction Entry	.273
Literal Origin Entry	.273
Constant-Definition Entry	.274
Origin Entry	.275
USING Entry	.275
PRINT Entry	.276
SET Entry	.276
Alignment Specification Entry	.277
Diagnostic Message Entry	.277
MNOTE* Entry	.278
TITLE Entry	.278
END Entry	.279
General Format for LOF Entry	.279
Global Section Macro Chain (GSM)	.280
Macro Name Dictionary	.281
Operation Code Table	.281
Machine Operations Requirements Table	.282
Using-Register Tables	.283
Macro Level Dictionary (Temporary Dictionary)	.284
Item Types	.285
Comments	.291
Source Line Storage Control	.292
Pseudo-Dictionary Item for Current Location Counter	.292
Constant Item Format	.293
Virtual Memory Management Table (VMTABLE)	.293
Source Program Listing	.294
Symbol Table Listing	.295
Cross-Reference Listing	.296
Internal Symbol Dictionary (ISD) Listing	.297

Program Module Dictionary (PMD) Listing297
Object Program Listing298
Internal Symbol Dictionary (ISD)300
Heading300
Section Name Table300
Using Tables300
Symbol Table300
Program Module Dictionary (PMD)304
PMD Heading304
Control Section Dictionary (CSD)307
CSD Heading307
Definition Table309
Reference Table309
Relocation Dictionary (RLD)310
Modifier Pointer310
Modifier310
RLD for Complex Definitions311
RLD for Text External Reference311
RLD for Text Internal Reference311
Virtual Memory Page Table (VMPT)311
 SECTION 13: VIRTUAL MEMORY MANAGEMENT312
Purpose of Virtual Memory Management Routines312
How Virtual Memory Management Works312
Routines312
VMGET -- Get VM Working Storage (CEVGM)312
VMFREE -- Free VM Working Storage (CEVFM)313
VMCLEAN -- Assembler Cleanup (CEVCU)313
Changing Storage Request Constants314
Caution314
Overflow Diagnosis314
 APPENDIX A: ASSEMBLER REGISTER USAGE316
 APPENDIX B: RELATIONSHIP OF DOCUMENTATION MODULES TO ASSEMBLY MODULES317
 APPENDIX C: ASSEMBLER LIMITATIONS319
Object Program319
PMD319
Text319
ISD319
Source Statements319
Macros319
Maximum Statement Length319
 APPENDIX D: ACRONYMS320
 APPENDIX E: LIST OF MAJOR TABLES AND WORK AREAS REFERENCED BY ASSEMBLER ROUTINES335
 INDEX338

ILLUSTRATIONS

Figure 1. Relationship of TSS/360 assembler with outside world . . . 1
Figure 2. TSS/360 assembler interface with LPC (to phase level only) . . . 2
Figure 3. Overview of entire assembler function . . . 6
Figure 4. LPC and assembler interface control flow . . . 7
Figure 5. Main work areas . . . 8
Figure 6. Overview of Phase I function . . . 10
Figure 7. Overview of Phase IIA function . . . 15
Figure 8. Overview of Phase IIB function . . . 17
Figure 9. Overview of Phase IIC function . . . 21
Figure 10. Overview of Phase III function . . . 22
Figure 11. Overview of Phase IV function . . . 26
Figure 12. Assembler function for machine instructions . . . 29
Figure 13. Assembler function for macro instructions . . . 30
Figure 14. Assembler function for assembler instructions . . . 31
Figure 15. LPC calls and assembler phase control flow . . . 32
Figure 16. Phase I routine relationships . . . 37
Figure 17. Waiting stack format . . . 75
Figure 18. Diagnostic text locator entry format . . . 80
Figure 19. Phase IIA routine relationships . . . 84
Figure 20. Phase IIB routine relationships . . . 88
Figure 21. Phase IIC routine relationships . . . 95
Figure 22. Phase III routine relationships . . . 100
Figure 23. Phase IIV routine relationships . . . 112
Figure 24. Cross-reference definition format . . . 113
Figure 25. Reference item format . . . 114
Figure 26. Absolute value item (EQU) . . . 261
Figure 27. Relocatable value item (DC, DS, CXD) . . . 261
Figure 28. Relocatable value item (machine instructions, CCW, LTORG, EQU) . . . 262
Figure 29. DXD item . . . 263
Figure 30. Complex value item (EQU) . . . 263
Figure 31. External name item (EXTRN) . . . 264
Figure 32. External name item (V-type address constant) . . . 265
Figure 33. Control section item (CSECT, DSECT, COM, START, PSECT) . . . 265
Figure 34. Entry trailer item . . . 266
Figure 35. Literal item . . . 267
Figure 36. Literal trailer item . . . 267
Figure 37. Transitive item . . . 268
Figure 38. Subscripted global arithmetic item . . . 269
Figure 39. Subscript trailer for subscripted global arithmetic item . . . 269
Figure 40. Unsubscripted global arithmetic item . . . 270
Figure 41. Subscripted global boolean item . . . 270
Figure 42. Unsubscripted global boolean item . . . 270
Figure 43. Subscripted global character item . . . 271
Figure 44. Trailer item for subscripted global trailer item . . . 271
Figure 45. Unsubscripted global character item . . . 271
Figure 46. Machine operation entry . . . 272
Figure 47. Macro instruction entry . . . 273
Figure 48. Literal origin entry . . . 273
Figure 49. Constant-definition entry . . . 274
Figure 50. Origin entry . . . 275
Figure 51. USING entry . . . 275
Figure 52. PRINT entry . . . 276
Figure 53. SET entry . . . 276
Figure 54. Alignment specification entry . . . 277
Figure 55. Diagnostic message entry . . . 278
Figure 56. MNOTE* entry . . . 278
Figure 57. TITLE entry . . . 279
Figure 58. END entry . . . 279
Figure 59. General format for LOF entry . . . 280
Figure 60. GSM entry format . . . 281
Figure 61. Macro name dictionary item . . . 281

Figure 62.	Item format for operation code table entry282
Figure 63.	Entry byte format283
Figure 64.	Using-register table format283
Figure 65.	Layout of macro level dictionary285
Figure 66.	&SYSLIST item286
Figure 67.	&SYSNDX item286
Figure 68.	&SYSECT item287
Figure 69.	&SYSPSCT item287
Figure 70.	&SYSSTYP item287
Figure 71.	Parameter item (temporary dictionary)288
Figure 72.	Sequence symbol item289
Figure 73.	Subscripted LCLA item289
Figure 74.	Unscripted LCLA item289
Figure 75.	Subscripted LCLB item290
Figure 76.	Unsubscripted LCLB item290
Figure 77.	Subscripted LCLC item290
Figure 78.	Unsubscripted LCLC item291
Figure 79.	GBLA, GBLB or GBLC item in macro level dictionary291
Figure 80.	Source statement control information format292
Figure 81.	Simulated item for location counter references292
Figure 82.	Constant item (address constant)293
Figure 83.	Constant item (other than address constants)293
Figure 84.	Contents of VMGOTTEN block294
Figure 85.	Contents of VMASSIGN and VMFREED blocks294
Figure 86.	Contents of VMENTRYS block294
Figure 87.	Source program listing format294
Figure 88.	Symbol table listing format295
Figure 89.	Cross-reference listing format296
Figure 90.	ISD Listing Format297
Figure 91.	Program module dictionary listing format299
Figure 92.	Listing format for constants301
Figure 93.	Listing format for machine and assembler instructions302
Figure 94.	Assembler internal symbol dictionary303
Figure 95.	Program module dictionary entry format305

Table 1.	LPC call to AC	32
Table 2.	Assembler control decision table	32
Table 3.	Phase I decision table (part	39
Table 4.	Standard variable information table	81
Table 5.	Phase IIA decision table	84
Table 6.	Phase IIB decision table	89
Table 7.	Phase IIC decision table	95
Table 8.	Phase III decision table	100
Table 9.	Phase IV decision table	113
Table 10.	Directive code assignments	282
Table 11.	Machine instruction directive codes	282
Table 12.	Virtual storage request constants	315

CHARTS

Chart AA.	AC (assembler control) - CEVAC117
Chart AB.	PHASE I (Phase I master control) - CEVPA118
Chart AC.	STAN (statement analyzer) - CEVST119
Chart AD.	REED (obtain next source statement) - CEVRD123
Chart AE.	GETOP (collect and identify operation code) - CEVGP125
Chart AF.	SUBOP (substitute into operation field) - CEVSP126
Chart AG.	CATOP (string substitution control) - CEVCP127
Chart AH.	MIP (machine instruction operand scan) - CEVMP128
Chart AI.	BASCAN (basic source language scan) - CEVBS131
Chart AJ.	AGO/AIF (AGO/AIF instruction scan) - CEVGO134
Chart AK.	ANOP and CCW (ANOP and CCW instruction scan) - CEVAN and CEVCW135
Chart AL.	CNOP and CXD (CNOP and CXD instruction scan) - CEVCN and CEVCX136
Chart AM.	SECT (control section instruction scan) - CEVCT137
Chart AN.	COPY (COPY instruction processor) - CEVCY138
Chart AO.	DC/DS (DC/DS instruction scan) - CEVDC139
Chart AP.	EJECT and END (EJECT and END instruction scan) - CEVEJ and CEVND140
Chart AQ.	ENTRY and EQU (ENTRY and EQU instruction scan) - CEVEY and CEVQU141
Chart AR.	EXTRN (EXTRN instruction operand scan) - CEVXN143
Chart AS.	GBLx/LCLx (global/local symbol instruction scan) - CEVGL144
Chart AT.	ICTL (ICTL instruction scan) - CEVIC145
Chart AU.	ISEQ and LTORG (ISEQ and LTORG instruction scan) - CEVIQ and CEVLG146
Chart AV.	MACRO and MEND/MEXIT (MACRO and MEND/MEXIT instruction scan) - CEVMC and CEVMX147
Chart AW.	MNOTE (MNOTE instruction scan) - CEVMN148
Chart BA.	ORG and PRINT (ORG and PRINT instruction scan) - CEVRG and CEVPR149
Chart BB.	SETX (SET instruction scan) - CEVSE150
Chart BC.	SPACE and TITLE (SPACE and TITLE instruction scan) - CEVCE and CEVTI151
Chart BD.	USE/DROP (USING and DROP instruction scan) - CEVUD152
Chart BE.	MACREF (macro reference processor) - CEVRF153
Chart BF.	MACDEF (macro definition processor) - CEVDF155
Chart BG.	CSCAN (constant scan) - CEVCS156
Chart BH.	SSCAN (string substitution scan) - CEVSS160
Chart BI.	EVAL (expression evaluator) - CEVEV164
Chart BJ.	PSCAN (parameter item analyzer) - CEVPS172
Chart BK.	EBIN and EDEC (binary and decimal self-defining term generator) - CEVGB and CEVGD173
Chart BL.	EHEX and ECHAR (hexadecimal and character self-defining term generator) - CEVGH and CEVGC174
Chart BM.	SLIT (scan for literal operand) - CEVSL175
Chart BN.	DLPM (dictionary lookup and put) - CEVLP176
Chart BO.	DEFSYM (define location symbol) - CEVSY177
Chart BP.	DIAG (diagnostic message processor) - CEVDX178
Chart BQ.	DLKT and DPUT (lookup and put in temporary dictionary item) - CEVTK and CEVTP180
Chart BR.	MACLKT and MACPUT (macro dictionary lookup and put) - CEVLM and CEVTM181
Chart BS.	DLKM (main dictionary lookup) - CEVKM182
Chart BT.	PHASE IIA (Phase IIA control) - CEVPB183
Chart BU.	PARAMAC (macro parameter processor) - CEVPM184
Chart BV.	PHASE IIB (Phase IIB control) - CEVPC187
Chart BW.	ORIGIN (location counter reset) - CEVGN192
Chart CA.	POOLIT (literal pooling processor) - CEVPL193
Chart CB.	EQUATE (assign value to name) - CEVEQ194
Chart CC.	RESCON (resolve conditional alignment) - CEVRS196
Chart CD.	RESLIT (literal resolution processor) - CEVRL198

Chart CE.	PHASE IIC (Phase IIC control) - CEVPD199
Chart CF.	USET (USING table processor) - CEVUP200
Chart CG.	DRSET (DROP table processor) - CEVDR201
Chart CH.	PHASE III (phase III control) - CEVPE202
Chart CI.	SLLS (source listing processor) - CEVSX208
Chart CJ.	GATEW (interface with VISAM PUT macro) - CEVGW210
Chart CK.	ENDPR (module entry point processor) - CEVEP211
Chart CL.	MOPR (Phase III machine operation processor) - CEVMO212
Chart CM.	GETVAL (obtain relocatable value) - CEVGV223
Chart CN.	USEVAL (compute using register) - CEVUV224
Chart CO.	LIST (object program listing) - CEVLS225
Chart CP.	CCWXTX (Phase III CCW instruction processor) - CEVCC231
Chart CQ.	PUTVAL (relocatable output value processor) - CEVPV233
Chart CR.	DCTXT (Phase III constant processor) - CEVDP235
Chart CS.	ADCON (address constant processor) - CEVAD238
Chart CT.	LITXT (Phase III literal pooling processor) - CEVLT242
Chart CU.	CSDPR (CSD processor) - CEVCD243
Chart CV.	PHASE IV (Phase IV control) - CEVPF246
Chart CW.	XREF (cross-reference listing processor) - CEVXF247
Chart DA.	STED (symbol table editor) - CEVSR248
Chart DB.	ISDPR (ISD processor) - CEVSD249
Chart DC.	PMDLS (program module dictionary listing processor) - CEVMD250
Chart DD.	ISDSA (ISD list processor) - CEVSA255
Chart EA.	VMGET (get VM working storage) - CEVGM256
Chart EB.	VMFREE (free VM working storage) - CEVFM257
Chart EC.	VMCLEAN (assembler cleanup) - CEVCU259

PURPOSE OF THE TSS/360 ASSEMBLER PROGRAM

The purpose of the TSS/360 assembler program is to produce, from source programs written in the assembler language, machine language programs in a format suitable for operation under the time sharing system. Outputs from the assembler program are:

- Source program listing
- Program Module Dictionary
- Program Module Dictionary listing
- Cross-reference listing
- Symbol Table listing
- Internal Symbol Dictionary
- Internal Symbol Dictionary listing
- Object program listing
- Binary Text
- External Name List

SYSTEM ENVIRONMENT

The initial request by the user to secure the assembler is processed by the command analyzer and executor (CA&E), which calls the language processor control (LPC). The language processor control calls the assembler, whose modules, resident in Initial Virtual Memory, are linked during startup.

As shown in Figure 1, the assembler makes use of:

- Language processor control to supply user program source statements.
- Symbolic library service routines to secure library definitions.
- Data management services to process output list data sets.

The assembler is called by and exits to the language processor control (LPC). The GETLINE function of LPC receives source-language statements from a system input device and directs them to the assembler for processing. Conversely, the assembled program and diagnostic messages are routed from the assembler to the same system output device via the PUTDIAG function of LPC.

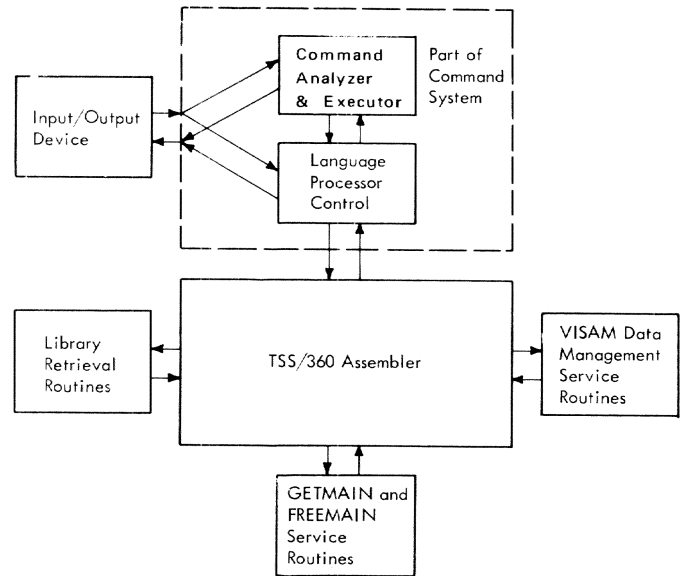


Figure 1. Relationship of TSS/360 assembler with outside world

In order to process COPY statements and macro instructions not defined by the user, the assembler searches user and system macro libraries. The library retrieval routines are used to accomplish this function.

An object program listing is automatically created for the user unless he stipulates otherwise. The source program listing, program module dictionary listing, cross-reference listing, symbol table listing, and the internal symbol dictionary listing must be requested by the user in his ASM command. Both the conversational and nonconversational user can choose between having the selected listings printed immediately on SYSOUT, or having them stored in a VISAM list data set. The default for conversational is a listing data set; for nonconversational the default is SYSOUT. If entered in a VISAM data set, the user's requested listings become members of a generation data group containing two generations. The generation data group is established the first time the module name is encountered. The most current listing data set (relative 0) and the last previous listing data set (relative -1) are always maintained. PRINT LIST.module-name.generation-number must be issued by the user when he desires the data sets to be printed.

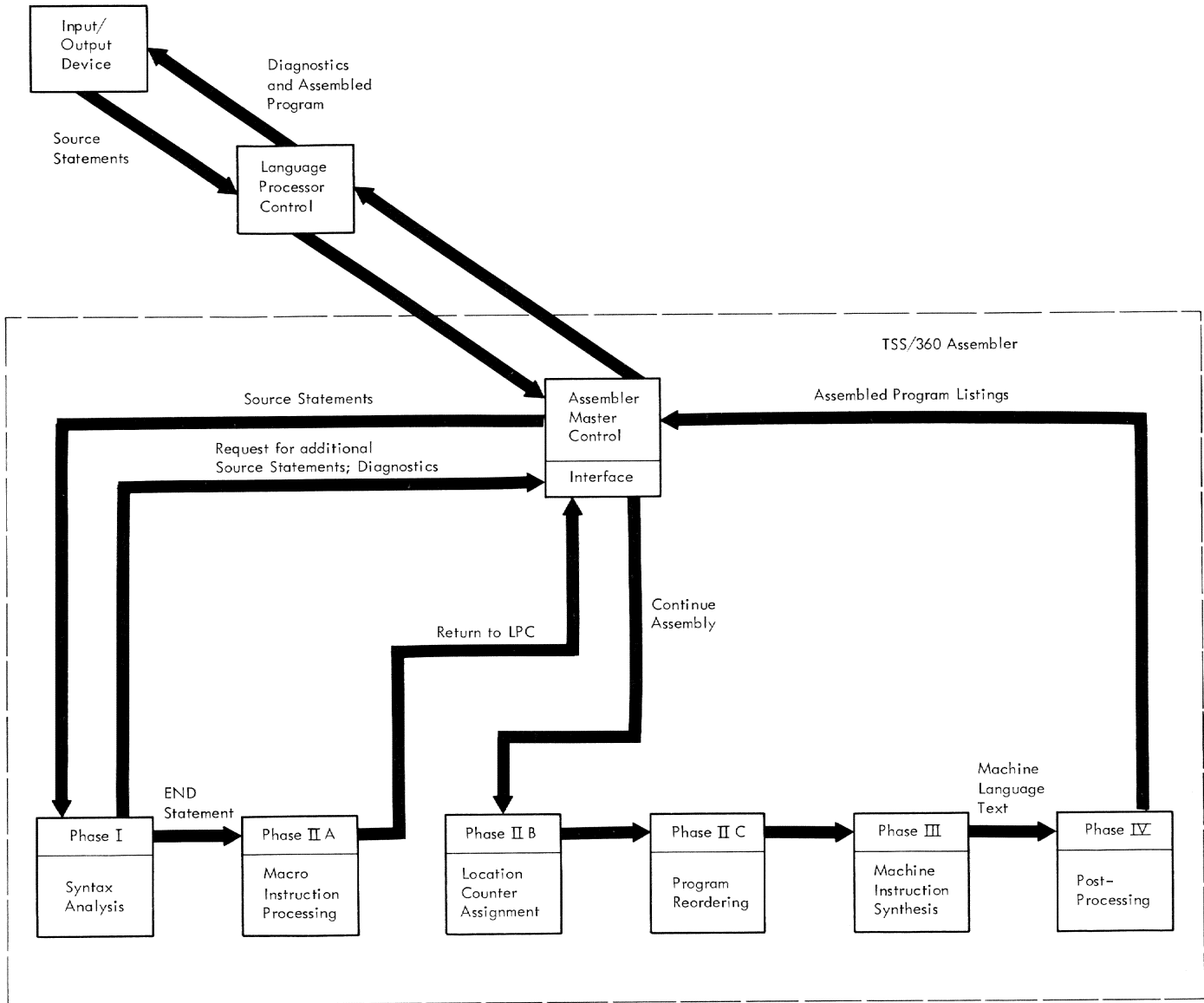


Figure 2. TSS/360 assembler interface with LPC (to phase level only)

Virtual storage dynamically acquired by the assembler is secured by the GETMAIN macro and released by the FREEMAIN macro. These macros are issued by special virtual memory management routines.

ORGANIZATION AND OVERALL FUNCTION OF THE ASSEMBLER

As shown in Figure 2, the assembler is divided into four major components or phases, plus an assembler control module which interfaces with LPC.

The principal function of any assembler is to translate computer instructions written in a symbolic language into the more

abstract, numeric language of the computer itself. This is accomplished principally by allowing alphameric symbols of the programmer's choice to represent the numerically addressed storage locations in the computer. The assembler's primary task is to determine which symbols have been defined, according to the rules of the assembler language, assign a corresponding machine-language value to the symbol, and to substitute the machine-language value whenever the symbol is used in the construction of a machine-language instruction.

In addition to this principal function, most assemblers also:

- Provide for the specification of numeric and alphameric data constants.
- Permit one symbol to be defined in terms of others.
- Recognize a vocabulary of control statements that apply to the assembly process itself (rather than the machine program under construction).
- Allow predefined sequences of source language statements to be generated and modified through use of a higher-level, machine-independent language (a "macro instruction" language).

The System/360 Assembler language contains all the above features; the method and order of their processing by the TSS/360 assembler is described in general terms below.

Syntax Analysis

In order for the assembler to interpret a statement without ambiguity, the programmer must follow certain rules in writing the source statement with regard to separation of fields, placement of symbols and delimiters, proper choice of mnemonic operation codes, and the like. The somewhat mechanical inspection of the source statement to determine whether the rules have been observed is generally called "syntax analysis," and is the first operation performed by the assembler on each statement. The analysis is achieved by a character-by-character scanning of the incoming statement; since this method of analysis is time consuming, the assembler usually converts the information that has been extracted from the statement into a more convenient internal form and places it in one of the various tables that are kept for this purpose. The principal tables are one that contains a condensed summary of each statement (the Logical Order File or LOF), and one that contains the name and characteristics of each programmer-defined symbol (the symbol table or "dictionary").

The definition of a symbol must be known to the assembler before it can construct a machine instruction that requires the value of the symbol. However, the rules of the language permit a symbol to be referred to before it is defined. If the assembler attempted to construct the machine-language program concurrently with syntax analysis, it would find itself frequently unable to do so for lack of information about symbols that had not yet been encountered. For this reason, construction of machine instructions is postponed until the entire source program has been syntactically analyzed and all symbols have been entered into the dictionary.

Macro Instruction Processing

A macro instruction is the invocation of a predefined sequence of source statements through use of a mnemonic operation code that has been declared for that purpose. The mnemonics of macro operations may be specified by the programmer himself, along with the sequence of statements that the operation represents, or, failing that, by the table of contents of a library of predefined macro operations that is present as part of the operating system. In either case, the assembler's dictionary of symbols cannot be considered complete until the sequences of statements represented by macro instructions have been syntactically analyzed.

In theory, macro instruction sequences may be processed either:

- Before the user's statements (by searching the source program only for macro instructions and by merging their expansion into the user's statements).
- Concurrently with the user's statements (by incorporating the expansion into the program as encountered).
- After the user's statements. The first method is used by other System/360 assemblers. The TSS/360 assembler, however, is committed to producing diagnostic messages for syntax errors for the benefit of a terminal user, and this requirement forces the assembler to process the user's statements first, as received.

Because system macros require the attributes of the user's symbols, and because there is no ordering rule (requiring the user's symbols to precede system macro calls), expansion of macros concurrently with the user's statements is also ruled out. Macros must be expanded by a second phase (Phase IIA) of the assembler after the user's statements have been syntactically analyzed.

Expansion of source statements from the predefined sequence in the macro definition involves the recognition of a class of symbols (variable symbols and parameters) which are independent of the symbols used in machine language statements. Since these symbols are used only temporarily (and may be used repetitively with different meanings), it is to the assembler's advantage to maintain them in a dictionary which is separate from the one used for machine-language symbols.

In addition, the expansion of one macro instruction frequently results in the invocation of some "inner" or "nested" macro

instruction. The rules of the macro language are such that it is desirable for the assembler to maintain a separate dictionary for each nested macro level. The rules of the macro language are also such that once the instructions have been generated for a given macro level, the dictionary for that level is no longer required and can be discarded, since symbols at each level are independent. For this reason, macro level dictionaries are constructed linearly in working storage, and maintained by push-down-stack logic.

Since the definitions of system macros are not part of the original user's source language input, they must be retrieved from a library and added to the source program at the appropriate time. Since library retrieval is time consuming, it is desirable to avoid retrieving a macro unnecessarily, and to retrieve each definition only once. This is achieved by performing library retrieval during the Phase IIA of the assembler; at this time those "nested" macro calls that are to be bypassed because of conditional assembly techniques are discarded, thus preventing their definitions from being unnecessarily retrieved. Moreover, a record is kept (in a special dictionary of macro names) whenever a definition is brought in; the definition is condensed into the internal form common to all statements, and need not be retrieved again should the macro instruction be reinvoked. This technique prevents multiple retrievals of the same definition.

Assignment of Location Counter Values

Once the additional statements generated by macro instructions have been incorporated into the source program, all possible and potential definitions of symbols are present in the dictionary. Before machine-instruction synthesis can begin, however, the (relative) machine address which each symbol represents must be determined. The value of the machine-address is arrived at by maintaining a location counter for each control section in the assembly. The counter is set to zero initially and is increased at each statement by the number of bytes of machine storage represented by the preceding instruction, constant, or storage reservation. Since macro instructions may generate instructions, constants, and storage reservations, the location counter cannot be assigned until macros have been expanded.

In those assemblers which expand macros first, the location counter can be assigned

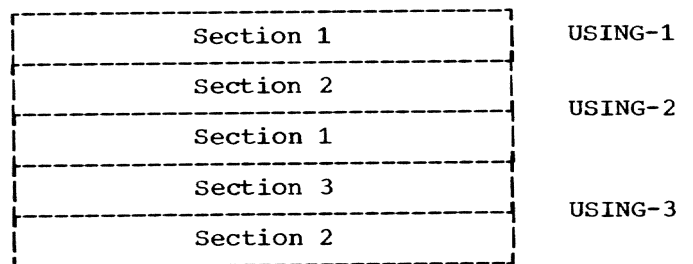
during syntax analysis; since the TSS/360 assembler defers macro expansion until Phase IIA (for the reasons noted above), location counter assignment is also deferred. For a better paging profile and ease of maintenance, Phase IIA is limited solely to macro expansion activity, and a separate phase, IIB, is used to perform the location counter assignment. As a byproduct of its principal activity, Phase IIB also resolves expressions that are dependent upon location counter values, and collects literal constants into literal pools and assigns location counter values to them.

Program Reordering

It is a requirement of TSS/360 object program modules that, to facilitate loading, all text and relocation information pertaining to a given control section be present contiguously in the object module. It is also a language rule that control sections may be written discontinuously in the source program, and that certain statements in the language (USING, DROP, LTOrg, PRINT, etc.) have effect over a range of statements in the original source order, irrespective of the number of different control sections represented by that range of statements.

The TSS/360 assembler is therefore faced with a reordering requirement. It must collect the scattered portions of a given control section, without losing the effect of certain statements that are control section independent. It is the function of Phase IIC to determine where each control section has been broken into discontinuities, and to prepare for each such break a table summarizing the effects of those statements that are independent of control section order. This analysis enables the machine instruction synthesis phase (Phase III) to collect the portions of a given control section and produce contiguous output text in the program module.

Graphically, Phase IIC transforms a program from:



to:

Section	} USING-1
1	} USING-2
Section	} USING-1
2	} USING-2
	} USING-3
Section 3	} USING-2
	} USING-3

Each entry point is to a location in the assembler control routine (CEVAC), from which control is transferred to the assembler location where the function is accomplished. Similarly, the two exits from the assembler to the LPC are also via the assembler control module.

The three entry points to the assembler control module are: to Phase I control (CEVPAA, Initiation), to Phase IIB control (CEVPAB, Continuation), and when abnormal termination is indicated (CEVPAZ, Early-end).

Machine Instruction Synthesis

When the reordering requirements have been resolved, the assembler is ready to begin the construction of machine-language instructions from their source language equivalents. Phase III performs this synthesis, working from a list of control sections in such a way that each control section, however discontinuously written, produces contiguous output text and relocation information for the loader. An expression evaluation routine, using information stored in the dictionary, resolves each machine-instruction operand to either a relocatable or absolute value. Appropriate text and relocation information is entered into the object module. Source and object program listings are a byproduct of this phase.

The two entry points of the LPC are: when the next line is desired, and when a diagnostic message is to be printed.

Figure 4 shows the flow of control between the LPC and the assembler.

Post-Processing

The user informs the LPC an assembly is requested, through the command language. The LPC then solicits the necessary operating parameters and enters the assembler at Phase I control for initialization. Command System PLM, GY28-2013, contains details of this operation.) Then the assembler enters the LPC to obtain the first source statement, and the LPC returns the call with the statement. The assembler processes the source statement and enters the LPC for the next statement. In an error-free assembly, this process is continued until an END statement is read, at which time entry is made from Phase I control directly to Phase IIA control.

When the assembly is complete and the object module has been produced, a series of post-processing routines may be called to operate upon the dictionary and other information left by preceding phases to produce sorted listings of the dictionary, cross-references to symbols, and analytical printouts of the various output modules. For convenience these routines are collected into Phase IV of the assembler.

Upon completion of Phase IIA, control is transferred to the LPC. If the assembly is in conversational mode, the LPC queries the terminal user whether to continue with the assembly, or correct the source program and restart.

ASSEMBLER FUNCTIONS

- If the user wishes to continue, Phase IIB control is entered and the assembler proceeds to completion without further conversational interaction.
- If the user makes corrections and wishes to restart, the LPC reenters Phase I control to restart the assembly.

Figure 3 is an overview which depicts the function and output of each of the four major assembler components. Note that Phase II is divided into three discrete parts, Phases IIA, IIB, and IIC.

The flow described above is altered automatically when the LPC determines a source line has been corrected, or the assembler discovers a source statement error.

A brief description of each phase function is given below and a more detailed description is given in succeeding chapters. For ease of understanding, the assembler control module is described last in this section.

When the assembler discovers an error in conversational mode, it calls the LPC with a diagnostic message, and LPC transmits the message to the system device (SYSOUT). LPC returns the call, and the assembler again calls the LPC for the next source statement.

Assembler Control Routine (Interface with LPC)

The assembler has three entry points from the language processor control (LPC).

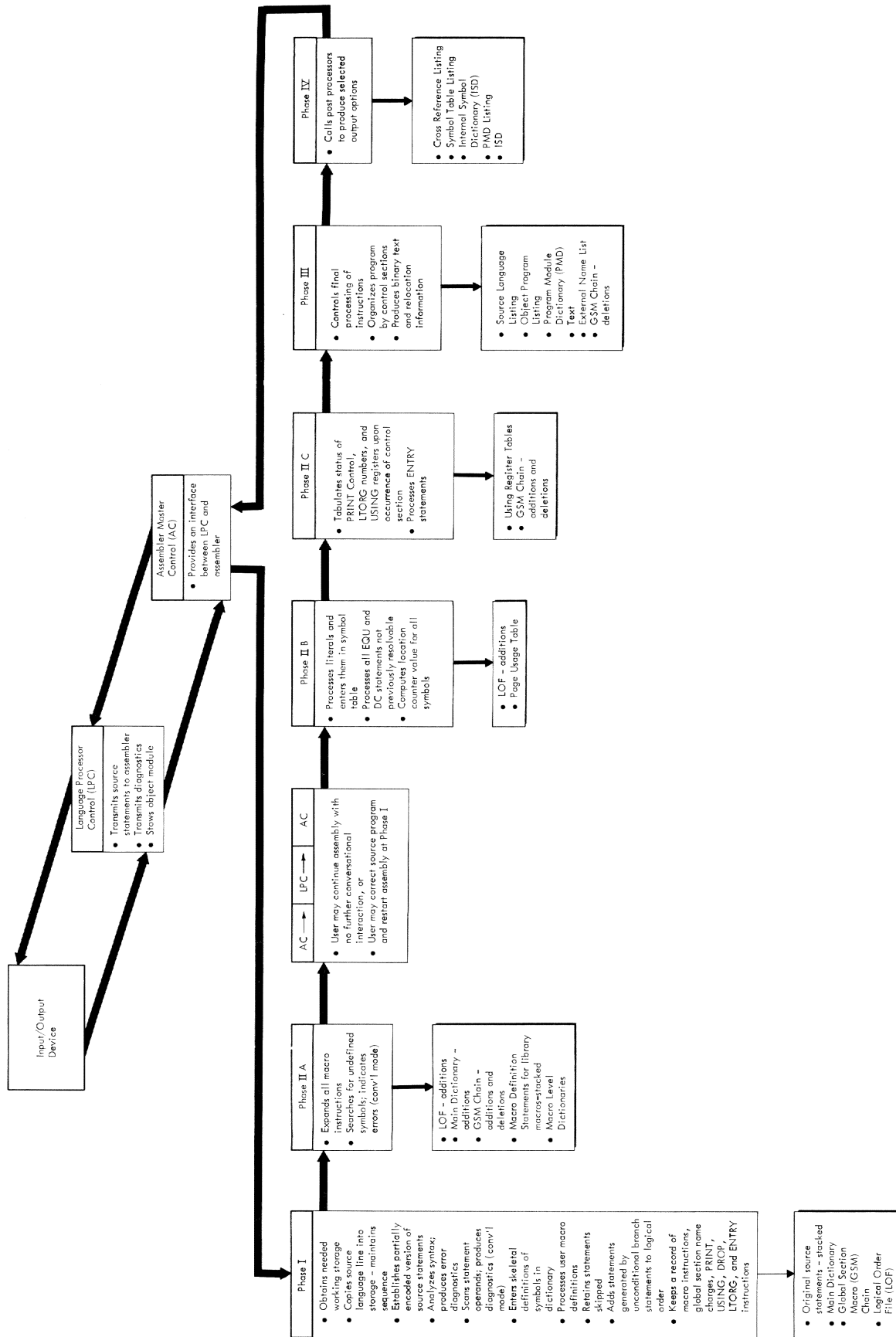
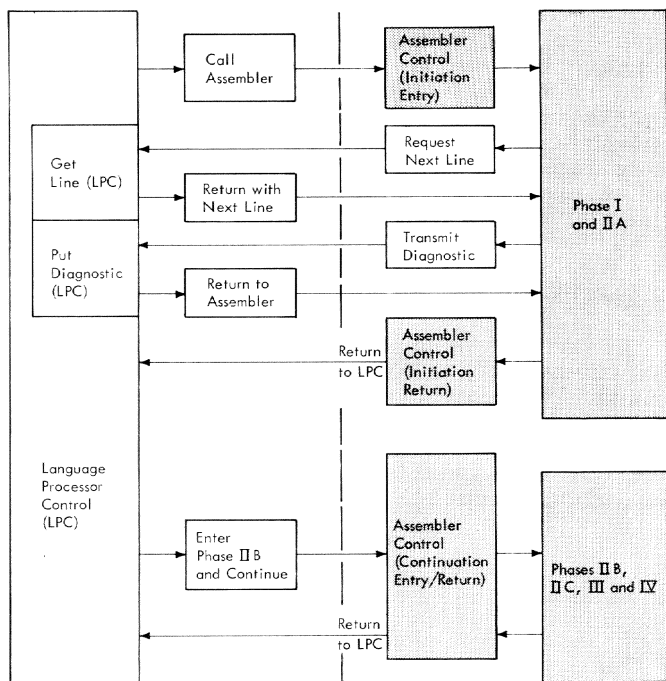


Figure 3. Overview of entire assembler function



• Abnormal Termination (Early End) not Shown

• Shaded Areas: Assembler

Figure 4. LPC and assembler interface control flow

If the LPC determines a source line has been corrected, it enters Phase I control with a special return code and the lowest line number to which corrections have been made. If the line number that LPC returns is greater than that of the next to last statement processed, the assembler processes the corrected statement and requests the next source statement from the LPC. If the line number is not greater than that of the next to last statement processed, the assembler reinitializes itself and starts over again by requesting the first source statement from the LPC.

If the assembly is interrupted by an Attention interruption, and a call is made to the LPC, the early-end entry of the assembler cleanup routine is called to release working space in virtual storage. Return is then made to the LPC.

USER VIRTUAL STORAGE REQUIRED BY ASSEMBLER

WORKING STORAGE AREAS

The TSS/360 assembler operates in and uses virtual storage as the communication medium for most of its input and output data. When the ASM command is given, LPC, the assembler, and all subprograms required by them are loaded into the user's virtual storage. In addition, the assembler

requests virtual storage dynamically for temporary and working storage.

Virtual storage is requested with the GETMAIN macro instruction. Assembler routines requiring working storage do not request it from the system directly; they go through the assembler's own management routines, which minimize the number of GETMAIN and FREEMAIN instructions issued. (This is discussed in detail in Section 13.)

The amount of virtual storage area requested by various assembler routines is controlled by constants in CSECT CEVPAS. Privileged system programmers (authority code 0) may change these constants to accommodate an exceptionally large assembly, such as assembling another language processor. (Refer to "Changing Storage Request Constants" in Section 13.)

If the assembler overflows its work areas, it will dynamically request additional virtual storage and continue processing if the storage is both available and addressable.

The three main work areas obtained dynamically by the assembler are outlined in Figure 5.

Work Area 1

The first page of Work Area 1 is reserved for pointers, work areas, and communication cells used between modules. During Phase IIA a part of Work Area 1 is used to store the macro level dictionaries needed for macro expansion. During Phase IIB the macro level dictionaries are overlaid with page usage tables. During Phase IIC using-register information overlays the page usage tables developed in Phase IIB. During Phase III the unused portion of Work Area 1 is used to hold sort keys for the cross-reference listing. The Phase III LIST routine uses the Operation Code Table as a work area for editing generated statements for the object listing.

Work Area 2

The first page of Work Area 2 also contains module cells, pointers, and communication cells used between modules. The second section is used for the main dictionary, the logical order file, and a secondary information list required for macro expansion (global-section-macro chain). Each type of information is structured as a list and is used as an open-ended working storage. During Phases I and IIA continued lines are carried in this area. During Phase III the previously unused portion of Work Area 2 is used as working storage for the construction of

various elements of the control section dictionaries.

Work Area 3

During Phase I Work Area 3 is used to hold incoming source statements for reference in later phases. During Phase IIA this area is used to hold statements generated by macro expansion.

PMD Text, ISD and External Names List Storage Areas

In addition to the three work areas described above, the assembler secures four additional virtual storage areas. The first area is for the PMD minus its associated text. Its size is equal to the number of binary text pages divided by

eight, plus two pages. The second area is equal to the number of pages required to contain the output binary text. The third area is for the ISD (if requested); it equals the number of pages in Work 2. The fourth area secured is for the external name list associated with the PMD. The locations of all four areas are passed to the LPC upon assembler completion.

CHARACTERISTICS OF ASSEMBLER ROUTINES

There are no hardware configuration requirements for any of the assembler routines. Most of the routines are reenterable, nonresident with respect to the system, nonprivileged, and closed; those that are not are specified as being so in the individual routine descriptions.

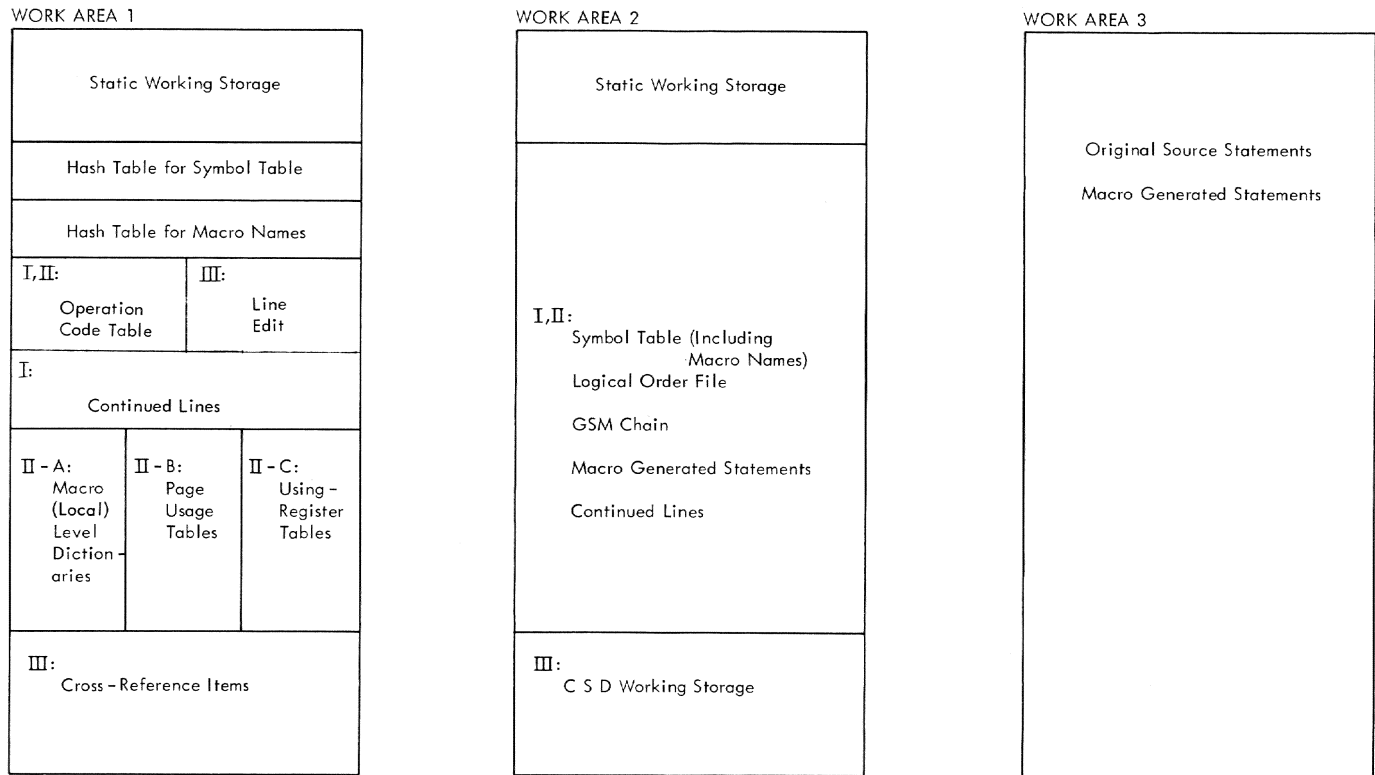


Figure 5. Main work areas

PHASE I FUNCTIONAL DESCRIPTION

Phase I is called by the language processor control program (LPC). It is the function of LPC to supply line-image items to the assembler, one at a time, upon request. The source language line is then copied by the assembler into its own working storage to facilitate references in subsequent phases and to serve as the input data for a source-language listing, when such is requested.

Since the assembly language permits transfers of assembler control and iteration over a set of source statements, the logical order of the assembly may be different from the sequential order. A principal function of Phase I is to establish a partially encoded version of the source statements (the logical order file) to establish the logical order of the assembly.

An overview of Phase I function is shown in Figure 6. The numbered paragraphs in the following description correspond to the numbered boxes in Figure 6.

1. Upon receiving control, Phase I calls VMGET to acquire two areas of virtual storage for its own working storage requirements. Initial and default values and beginning addresses for variable storage are inserted into the static portion of working storage. Static working storage is also modified as a result of the operating parameters transmitted by the LPC. Having established the source program data set as the current input source, control is transferred to the statement analyzer for the program to be processed.
2. The statement analyzer controls the processing of each source language statement in order by using a collection of specialized subroutines. It produces the symbol dictionary, the global-section-macro (GSM) chain, and the logical order file (LOF) from which Phase III produces the output program module. It has two modes of operation: normal and bypass. In the normal mode, source lines are obtained and processed to produce some change in the information compiled by the assembler to further the production of an object program. The bypass mode is initiated by the processing of an AGO

or true AIF command whose transfer point is a sequence symbol that is as yet undefined. In this mode, source lines are merely bypassed until a line containing the desired sequence symbol is encountered, at which point normal processing is resumed.

3. REED is called by the statement analyzer to obtain the next source statement. REED provides the interface with LPC to obtain source lines. It concatenates continuation lines to provide the statement analyzer with a continuous statement, performs sequence checking, and switches the source of input statements between LPC, macro definitions, and COPY-library statements, as required. REED obtains source lines directly from the language processor control (LPC) or from a library when obtaining a macro definition to satisfy a macro instruction. During Phase IIA the principal source of input is the macro expansion mechanism rather than the LPC.

Regardless of its origin, a source line may be in either keyboard or card image format and a source statement may comprise multiple source lines, through the statement continuation capabilities. In obtaining the next source statement, if REED encounters a source line that is continued, all the portions of the statement are combined into a single continuous line that is constructed in assembler working storage. REED is also responsible for performing and commenting diagnostically upon failures in the sequence check demanded by the prevailing ISEQ requirements.

REED provides the capability to furnish the conversational user with the ability to correct or delete the last source statement presented to the processor without incurring restart of the entire assembly. It records the internal status of the assembler as each source statement is completed. Thus, at any time prior to commencing the processing of the next statement, the effect of the current statement can be erased by replacing the current status information with the previous status, and by detaching from linkage chains any dictionary items constructed since the previous status was recorded.

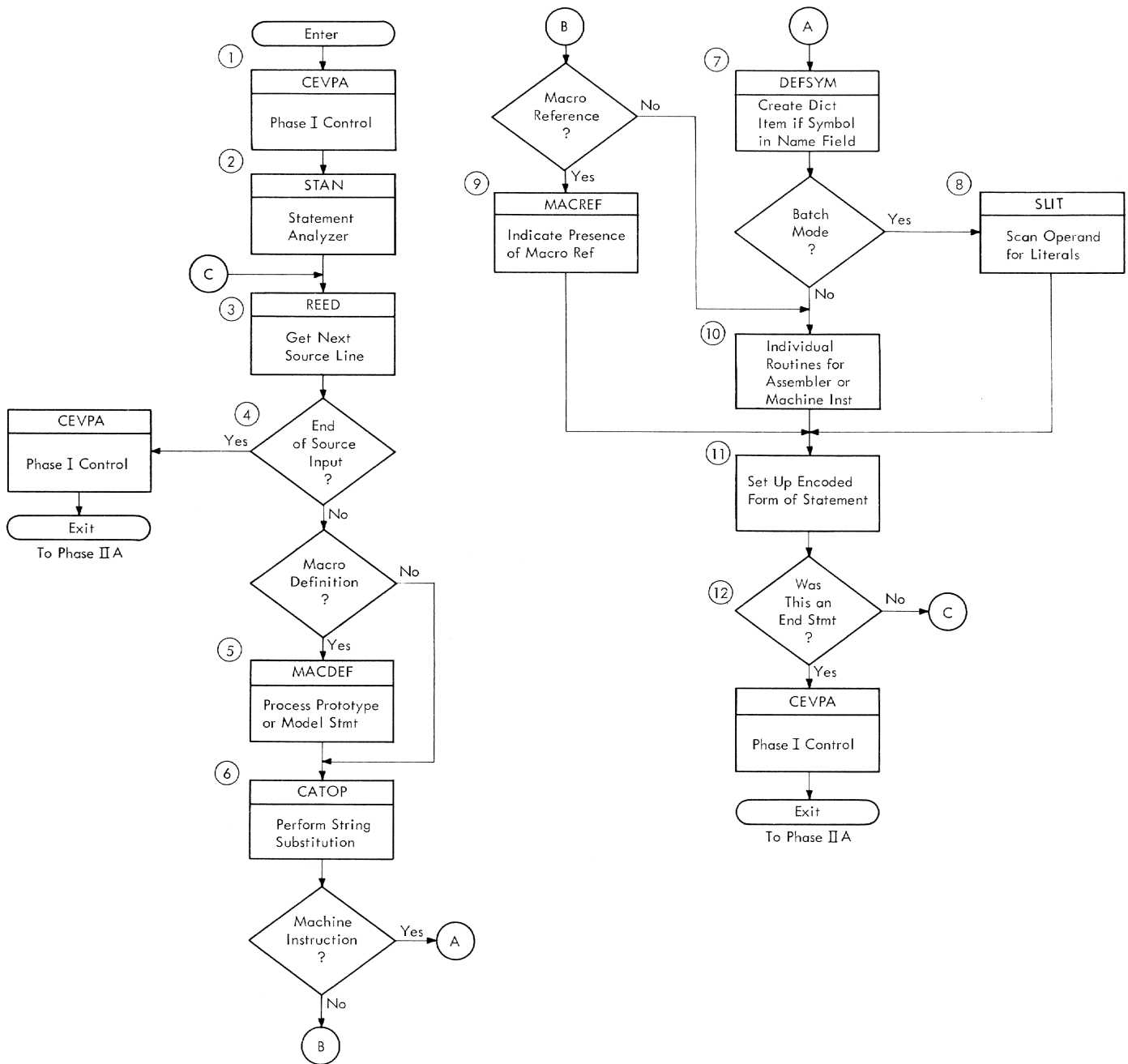


Figure 6. Overview of Phase I function

REED initializes the construction of a logical order file (LOF) entry, by setting the entry to zero. The LOF entry, which represents the encoded form of the statement, is built at a temporary location. REED calls GETOP, which determines the required length of the LOF entry, based upon the type of statement being processed. GETOP isolates the operation mnemonic (via SUBOP) and identifies it (by doing a binary search on the operation code table) as a machine operation code,

assembler instruction, user macro, or library macro. The directive code, which classifies the statement as to type of machine instruction or assembler mnemonic, and the operation code are placed in the LOF entry by GETOP.

4. If the end of source input in a pre-stored data set has been encountered, and no END statement was provided, the LPC will supply an END statement. A diagnostic is issued stating that the

END card is missing. The LPC will return to the assembler, which will process the END statement. Because the END statement signals the end of the phase, control is then passed to Phase IIA.

5. If the current statement is either a macro prototype statement or a model statement, the macro definition processor (MACDEF) is called. If the statement is a prototype, a macro name item is constructed in the main dictionary. The operation code is looked up in the operation code table and, if a match is found, a diagnostic is issued warning that an operation mnemonic has been redefined by a macro definition.

The redefinition indicator is turned on in the matching operation code table entry. The dictionary item for the macro name is completed by MACDEF by inserting the location of the LOF entry for the prototype line and the location of the prototype line itself. The former is used by the macro reference processor (MACREF) in initializing the REED subroutine to read the definition when the macro is expanded. The latter is used by the PARAMAC routine in Phase IIA to establish a temporary macro-level dictionary when the macro is expanded.

If the current statement is a model statement, a diagnostic will be issued if the operation code is ISEQ, ICTL, or END. COPY statements cause MACDEF to call the COPY subroutine, which reads in the library element and pushes down the input-source switch in REED so the subsequent statements originate from the library. Thus, copied statements become part of the macro definition and not part of the expansion.

6. All statements pass through the string substitution control routine (CATOP). This routine controls the type and amount of parameter and variable symbol substitution that is applied to the current source statement. It is called before the statement is delivered to the components of the statement analyzer for processing. Substitution will have been performed arbitrarily on the operation code field by SUBOP (via GETOP) prior to identification of the operation mnemonic. CATOP calls the string substitution scan routine (SSCAN) to perform string substitution on the name and operand fields. Whenever substitution actually results in character string replacement on a statement, a new version

of the statement reflecting the substitution is produced to replace the original line for all subsequent processing. After substitution, CATOP calls the basic scan routine (BASCAN) to analyze the contents of the name field. CATOP then determines the start of the operand field and posts the increment from the beginning of the statement in the current LOF entry.

7. All machine instructions pass through the define location symbol routine (DEFSYM). Its purpose is to construct and enter into the main dictionary a relocatable value item that represents the name field symbol (if present) of the current source statement. DEFSYM calls the main dictionary lookup and put routine (DLPM) to look up the symbol in the dictionary and construct a skeletal item. DEFSYM completes the skeletal dictionary item according to the type of the current operation code and reserves the space for the item. The location of the item is then entered into the current LOF entry.
8. If the current statement is a machine instruction, and the assembler is in nonconversational mode, the scan for literal operand routine (SLIT) is called in lieu of the complete operand field analysis routine. Its function is to scan the operand field to determine whether a literal operand (denoted by the character =) is present. If a literal is found, the location of the = in the source statement is added to the current logical order file entry.
9. If the current statement is a macro instruction, the macro reference processor (MACREF) is called. MACREF is responsible for indicating the presence of the macro instruction. An entry for the GSM chain is constructed to cause the expansion of the macro instruction in Phase IIA.
10. If the assembly is conversational, and the current statement is a machine instruction, the machine instruction operand scan (MIP) is called to scan the operand and check for valid operand fields and correct formatting.

If the assembly is in either batch or conversational mode, and the current statement is an assembler mnemonic, individual routines will be called for each mnemonic. These routines are described below.

AGO/AIF: The sequence symbol appearing in the statement is processed and the input source is reset to an earlier symbolic statement, if backward; or, if forward, a bypass mode is instituted.

ANOP: The name field is checked for the presence of a sequence symbol.

CCW: DEFSYM is first called to create a dictionary item if there is a symbol in the name field. The operand is examined for valid operand fields and correct format if the assembly is in conversational mode. Otherwise, SLIT is called to scan the operand for the presence of literals.

CNOP: If the assembly is in conversational mode, the operand fields are examined and checked for validity.

COM: Described under SECT.

COPY: The desired element is retrieved from the library and copied into working storage in the form of chained source lines. The input-source switch of REED is set to retrieve forthcoming statements from the copied stack.

CSECT: Described under SECT.

CXD: DEFSYM is first called to create a dictionary item if there is a valid symbol in the name field. Upon return from DEFSYM, the current logical order file is completed, and a constant item is constructed.

DC/DS: DEFSYM is first called to create a dictionary item if there is a symbol in the name field. The constant scan routine, CSCAN, is called to process the operand field and construct a constant item. Attributes are obtained from the constant item and posted in the current location symbol item, if there is one. If the end of the operand field was not reached by CSCAN, an additional logical order file entry is created and CSCAN called again. Thus, individual logical order file entries are constructed for each operand of a multiple operand statement.

DROP: Described under USE/DROP.

DSECT: Described under SECT.

EJECT: The only processing required at this time is a check to determine if the name field is blank.

END: An indicator is set to the effect that the END statement has been

encountered. If the assembly is in conversational mode, the operand field is examined and checked for validity.

ENTRY: The name field is checked for blanks or a sequence symbol, and a GSM entry is constructed. If the assembly is in conversational mode, the basic scan routine (BASCAN) is called to collect and examine each operand field.

EQU: At this time only an EQU in which the first operand expression yields an absolute or complex value can be fully processed. An absolute or complex value item is created in the dictionary, and the length and type attribute fields are evaluated and processed. Other operand types cause a transitive item to be created and an indicator to be set in the logical order file entry to demand attention in Phase IIB.

EXTRN: Each symbol in the operand is collected and an external name item constructed for each.

GBLX/LCLX: If the statement occurs within a macro, an item is constructed in the temporary dictionary; if the operation is global, an item is also constructed in the main dictionary.

If the statement occurs outside a macro, an item is constructed in the main dictionary. If a subscript is present, it is checked for validity and its value inserted into the dictionary item. If the operation is global and is not Phase IIA, a GSM entry is constructed.

ICTL: Checks are made to determine if the statement is the first source program statement, and if there is only one ICTL in the assembly. The routine checks if each operand field is valid.

ISEQ: The operand fields are examined for validity and indicators set with the new values.

LCLX: Described under GBLX/LCLX.

LTORG: DEFSYM is called to create a dictionary item if a symbol is in the name field. A GSM entry is created and the logical order file entry is flagged for special attention in Phase IIB.

MACRO: After the statement is checked for syntax, the macro definition switch is set to 1 and control is returned to the statement analyzer.

MEND: Processed by MEND/MEXIT routine. If the macro definition mode is set, it is canceled and an immediate return is made. This condition prevails during the processing of macro definitions. If the macro definition mode is not set, MEND executes identically with MEXIT.

MEXIT: Processed by MEND/MEXIT routine. The space occupied by the current macro level dictionary is reclaimed. The macro level is reduced by one, and the location of the logical order file entry for the statement at which processing stopped on the preceding macro level is reinstated in the REED input switch. If the macro level has been reduced to zero, the REED input switch is popped up to its previous mode.

MNOTE: The first operand is examined and, if it is an asterisk, the character string that follows is treated as a comment. Otherwise, the character string is considered a diagnostic message and causes a special call to the diagnostic processor.

ORG: The logical order file entry is flagged for the attention of Phase IIB. If the assembly is conversational, the operand field is examined for validity.

PRINT: An entry is made in the GSM chain so that the effect desired by the source programmer can be produced by subsequent processing by control section. The operand field is tested for the legitimacy of its contents.

PSECT: Described under SECT.

PUNCH: The statement is allowed only to maintain compatibility with OS/360 and is made commentary.

REPRO: The instruction produces listing only; the following statement will also be treated as commentary.

SECT: The symbol in the name field or blank denoting blank COMMON (and binary zero denoting a blank CSECT) is used as the basis for a main dictionary lookup and, if the symbol is not in the dictionary, a section-name item will be created for it. An entry is inserted into the GSM chain and, for control sections other than DSECT or START, the operand field is examined for attribute declarations. The operand of a START is processed like that of an ORG statement, and an

ORG logical order file entry is generated following the START entry. The operand of a DSECT is not examined.

SETX: The symbol in the name field is looked up in the main or current macro level dictionary to verify if an item exists for it. The operand expression is now evaluated, and the value is posted in the item. If a global symbol is being set, and if in Phase I, a GSM entry is made for the statement.

SPACE: If the assembly is conversational, the operand field is examined for validity.

START: Described under SECT.

TITLE: The name field is saved for later use in card identification. The character string in the operand is saved for later use in printing the assembly listing.

USE/DROP: For USING instructions, the first operand field is evaluated to see that it is valid. For either USING or DROP, the register designations are examined for validity. A GSM entry is created for either instruction. No operand examination is completed if the assembly is in batch mode.

11. If the current statement implies source code, and a control section has not been declared, a logical order file entry will be set up for an implied CSECT. A GSM chain entry will also be constructed for the implied CSECT. After all generated entries have been constructed in working storage, the logical order file entry for the current statement is moved from its temporary location into working storage and the previous entry linked to the current one.
12. If the current statement is an END, exit is made from the statement analyzer to the Phase I control, which then passes control to Phase IIA.

PHASE IIA FUNCTIONAL DESCRIPTION

Phase IIA is responsible for the expansion of macro instructions and, when required, the retrieval of system macro definitions from the library. During Phase I a record is maintained for all macro instruction source statements; Phase IIA completes the processing of those statements.

Macro statement generation is accomplished by substituting the character-string values of the current arguments for the corresponding parameters in the definition. The macro definition statements remain in the sequenced source statement area in the virtual storage of the assembler. The source statements generated by macro instructions are also retained in the virtual storage of the assembler; they do not become part of the set of sequenced statements. When the generation of each new symbolic statement is complete, the statement is subjected to standard Phase I processing and is assembled as if it had been part of the user's original source program. Most of the processing routines which were present in Phase I are present in Phase IIA also; however, Phase IIA acts as an internal replacement for LPC in determining the order and origin of the source statements.

As a corollary to the processing of macros, Phase IIA must reevaluate statements that affect global variable symbols and must maintain a record of control section and print status changes. Before concluding, Phase IIA also presents global diagnostic messages to the conversational user and calls LPC to determine whether to continue the assembly.

An overview of Phase IIA function is shown in Figure 7. The numbered paragraphs in the following description correspond to the numbered boxes in the figure.

1. Activity in Phase IIA is controlled by the entries in the GSM chain. This chain is prepared during Phase I and contains entries for each macro instruction, GBLx instruction, SET statement involving a global symbol, PRINT, and change of control section. Other entries in the GSM chain are not pertinent to Phase IIA.
2. If a control section GSM is encountered, it is necessary to retrieve the location of the section name item in the main dictionary and point the current control section indicator to this item. This pointer may also be updated by the control section processor (SECT) if a control section statement occurs during a macro expansion. The section name item is used to establish the various values for &SYSECT as macro expansion proceeds.
3. If a GSM entry representing a GBL statement at the user level is encountered, it is necessary to reprocess the statement to ensure synchronization of user-defined global variable symbols with the macros expanded during this phase. At the first rede-

claration of each symbol the initial value of the item is reset to the null state.

If a GSM entry representing a SET statement at the user level is encountered, it is also necessary to reprocess the statement to maintain the synchronization of global variable symbols established above. The value of the global symbol originally obtained in Phase I is retained in the logical order file and is reinstated by Phase IIA.

The basic scan routine (BASCAN) is called for either a GBL or SET statement to preset pointers for the Phase I routine for the instruction. The appropriate GBL or SET routine in Phase I is then called. The GSM entry is removed from the chain, and processing continues with the next GSM entry.

4. Encountering a GSM entry for a PRINT instruction causes the print status to be replaced with that carried in the LOF entry pointed to by the PRINT GSM.
5. When a GSM entry representing a macro instruction is encountered, Phase IIA control calls upon the statement analyzer to process the macro instruction. The statement analyzer, being phase-conscious, calls the macro reference processor, MACREF, which determines if the macro is at the user level (macro definition exists in storage).
6. If the macro definition is not in storage, the macro is a library macro. The library service routines, CEVMLA and CEVMLB, are employed by MACREF to search the library for the desired macro and to retrieve the lines of the macro definition. Lines are retrieved and are linked together in working storage. A mode switch is set so that the REED routine can process the statements from the library instead of in normal mode.
7. The statement analyzer is entered at a special entry point (CEVST1) from MACREF to call the REED routine to initiate construction of logical order file entries for the definition statements. After the statement analyzer processes the statements, the logical order file entries for them will be delinked from the main chain, but maintained for subsequent reference. The statement analyzer returns control to MACREF after processing the MEND statement of the library macro.

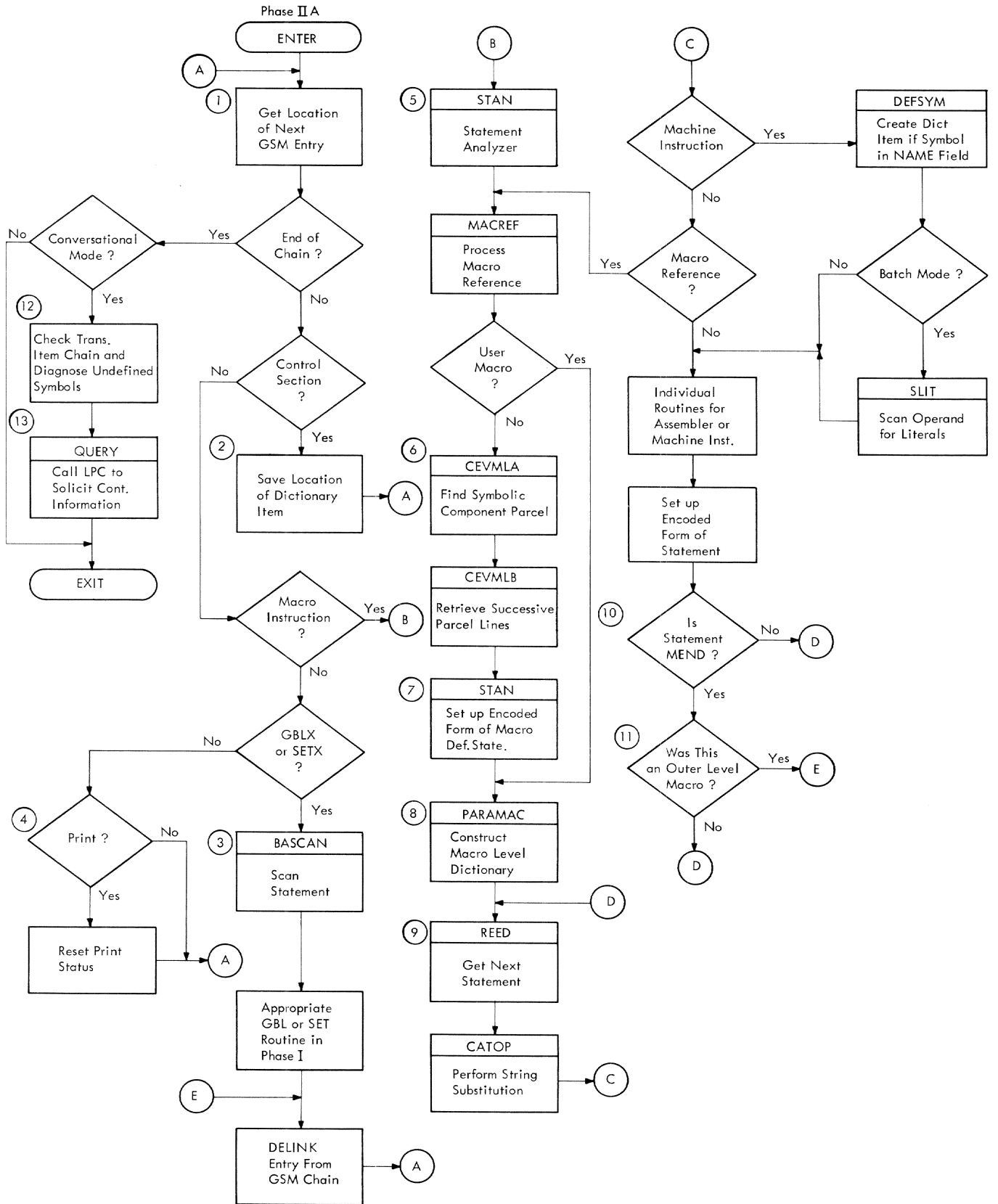


Figure 7. Overview of Phase IIA function

8. After it is ensured that the macro definition exists in storage, the macro parameter processor, PARAMAC, is called by MACREF to expand the macro. Temporary dictionaries are created by PARAMAC for each outer and inner macro instruction level. The symbolic parameters in the macro prototype statement and the corresponding positional operands or name field in the macro instruction are combined to form parameter items in the temporary dictionary. Each item is identified by the symbolic parameter, which is hashed and linked to an entry in the macro hash table for the current level. Each temporary dictionary contains the linkage and status information necessary to initiate an inner-macro expansion, to purge the temporary dictionary of an inner macro after expansion is complete, and to resume processing of the macro at the next higher level. The remainder of the dictionary at each level contains a reduced hash table followed by parameter items representing the system variables and symbolic parameters specified in the macro definition. PARAMAC returns control to MACREF which, in turn, returns control to the statement analyzer.
9. The statement analyzer calls REED to initialize processing for the next statement. The model statements are fed through the main routines of Phase I (that is, CATOP, DEFSYM, SLIT, and the appropriate individual routine) in the same way as user statements were fed through in Phase I. The logical order file entry for the generated statement is constructed in working storage and is flagged as representing a generated statement.
10. If the MEND for the outer level macro has not been encountered, processing continues as in 8.
11. When a MEND is encountered for the outer level macro, processing of the macro instruction is complete. The GSM entry for the macro instruction is removed from the chain and processing continues with the next GSM entry. If the macro with which the MEND is associated is not an outer level macro, processing continues as in 8.
12. When all GBL, SET, PRINT, control section, and macro instruction entries in the GSM chain have been processed, Phase IIA passes over the transitive items in the main dictionary and extracts all symbols that remain undefined if the assembly is in conversational mode. These symbols are represented in the dictionary by transitive

items that have not been completed by the insertion of the location of the matching definition. Diagnostic messages are produced for each symbol.

13. Control is given to the entry of LPC which solicits continuation information from the conversational user. If the user elects to continue, control returns to the assembler at "continuation" entry point, and assembly resumes with Phase IIB.

PHASE IIB FUNCTIONAL DESCRIPTION

At the conclusion of Phase IIA, the entire source program has been fully scanned once. It is the task of Phase IIB to organize the results of this initial scan so that the object text can be generated in a single pass over the internal representation of the program. The principal function of Phase IIB is to assign location counter values to symbols and literal constants.

The areas requiring resolution are:

Boundary Alignment: The generation phase requires space unused because of boundary adjustment to be claimed explicitly.

Literal Assignment: For each literal reference, the value and length of the constant is to be computed and duplicates are suppressed.

Literal Pooling: As dictated explicitly by LITORG statements or implicitly by the program end, literals are to be arranged by their length modulo 8 and assigned location counter values.

Symbol Definition: All definitions, unless erroneous, must now be capable of resolution by the assignment of a location counter or absolute value, as appropriate.

An overview of Phase IIB function is shown in Figure 8. The numbered paragraphs in the following description correspond to the numbered boxes in the figure.

1. Phase IIB makes a single pass over the logical order file; the processing that is performed depends upon the characteristics of the entry in the logical order file. The entries may be grouped into three categories: location counter adjustments, literal operands, and normal statements. Location counter adjustments may be further subdivided into changes of control section, literal origin statements, ORG statements, and conditional storage reservation statements (such as CNOP or DS statements).

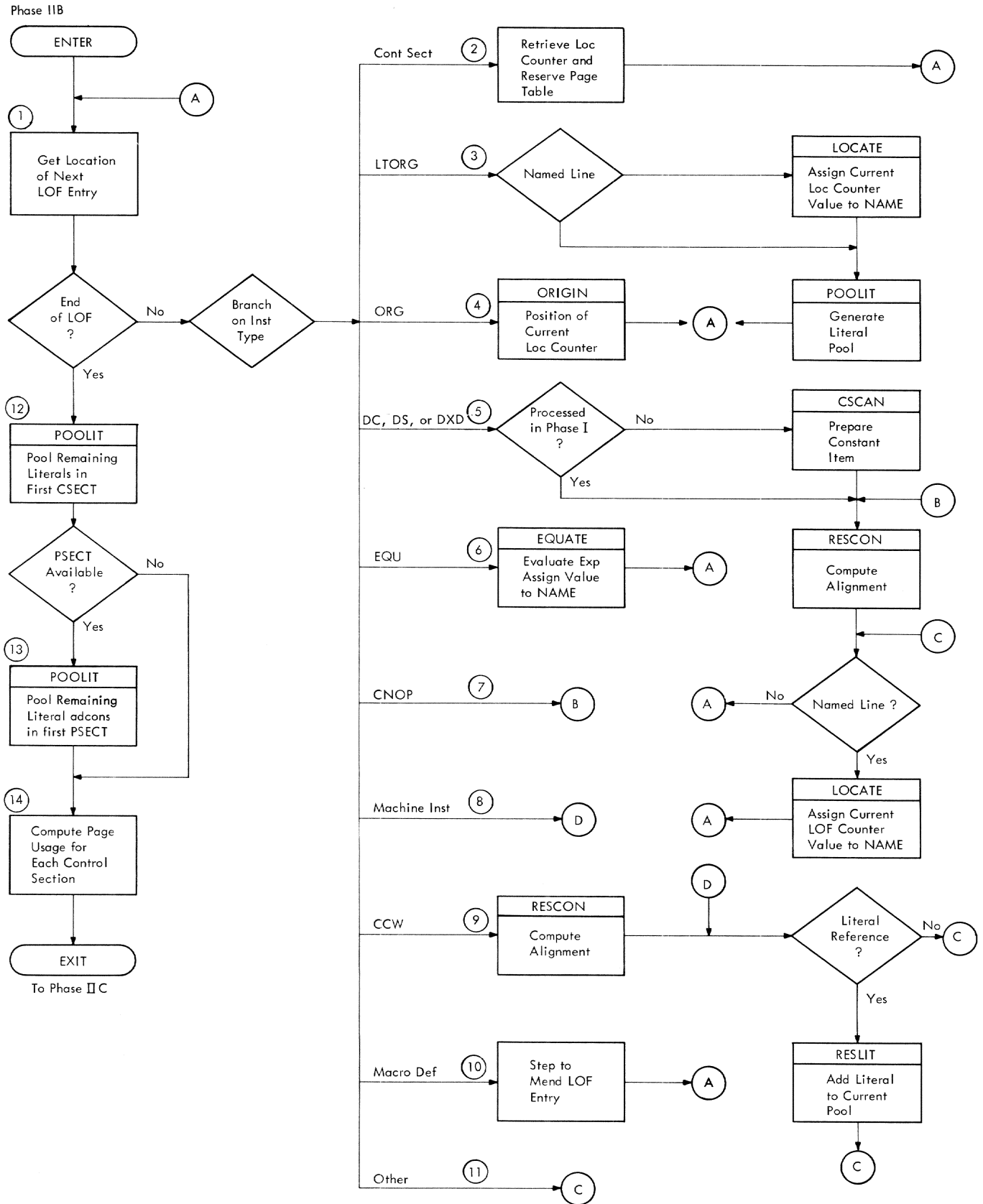


Figure 8. Overview of Phase IIB function

2. If the logical order file entry represents a control section entry, the following processing is performed.

Each control section within the assembly has its own individual location counter, for which two values are maintained: the current value as it exists for any given statement, and the highest value the counter has reached during the course of processing the control section. At any change of control section the current value of the location counter is saved (in the section name item in the dictionary). If this value exceeds the highest value previously saved, the highest value is also updated and saved. The current value of the location counter for the new section is retrieved and installed as the working counter for subsequent statements. The location of the section name item for the new section is also inserted in the current control section indicator.

For each occurrence of a new control section other than a blank COM or DSECT, a 512-byte "page usage" table is reserved in working storage. The table length provides one bit for each of 4096 pages allowed for a control section. Each time the location counter is incremented, and the incrementing instruction is other than a DS or ORG, a bit is set in the corresponding page usage table, indicating that the page represented by bits 8-19 of the current location counter contains text. If a statement will cause the location counter to exceed the limit of 4096 pages, that statement is made commentary, and the remaining statements (except the END statement) are also made commentary. The page table is initiated to zero at the time it is reserved, and its location is inserted in the section name item.

3. If the current logical order file entry represents a LTORG statement, the location counter is first aligned to a doubleword boundary. If the statement has a symbol in the name field, LOCATE will be called to assign the current location counter to the symbol. Next, POOLIT is called to generate the literal pool. The literals are chained in order of occurrence to a "first link" which is independent of the dictionary. POOLIT's function is to order the literals by length, assign location counter values to each literal, and to transfer the chain (reordered by ascending location) to the LTORG entry in the logical order file. If a PSECT is present in the

assembly, POOLIT excludes address constants from the pool, unless an override switch is set (indicating the absence of a PSECT) during Phase IIB initialization to force their inclusion.

The LTORG statements in a program are numbered in order of occurrence. Literals occurring between (or prior to the first) LTORG statements are identified as belonging to the LTORG number which is forthcoming.

4. If the current logical order file entry represents an ORG statement, ORIGIN is called to evaluate the operand of the ORG statement. Absolute values receive a diagnostic message but are then accepted as indicating a location counter setting relative to the current control section. Relocatable values must be simply relocatable and relative to the current control section. A null operand indicates that the location counter is to be set to the highest previously attained location counter value for the current control section.

If the value of the new origin is less than the current reading of the location counter, the current value is compared against the highest previously attained value (preserved in the section name item). If the current value is higher, it replaces the previous high value. If the current value is not higher, it is discarded. In either case, the new origin is instated as the current value of the location counter and placed in the logical order file entry for ease in listing in Phase III.

5. If the current logical order file entry represents a DC or DS instruction, a test must first be made to determine if a constant item was constructed for the statement in Phase I.

If the statement was incapable of resolution in Phase I because of lack of definitions for terms in the expressions for length, duplication, scale or exponent, the constant scan routine (CSCAN) is called to construct a constant item. If an item cannot be constructed, the statement is considered invalid and is treated as commentary. If a constant item was constructed in Phase I, Phase IIA, or Phase IIB, RESCON is called to resolve any conditional alignment. The type of alignment required is indicated in the constant item. RESCON aligns the working bit location counter to the proper boundary and, if the alignment amount

is nonzero, constructs an alignment LOF entry. This entry indicates the number of bits to skip. RESCON also inserts the alignment entry in the logical order file preceding the entry for the current DC or DS statement. This ensures that Phase III will make an identical adjustment to the location counter. The working byte location counter will be set to the truncated value of the bit counter.

If there is a symbol in the name field, LOCATE is called to assign the current location counter value to the name.

A DXD instruction is treated as a DS instruction with the following exceptions: an entry is made for the DXD on the Q REF chain, and the location counter is unchanged.

6. If the current logical order file entry represents an EQU statement, the EQUATE subroutine is used to determine the legitimacy and the value of the operands of the statement. EQUATE will be entered only if the value of an EQU statement was unobtainable during Phases I or IIA. The applicable type and length attributes of the value item are entered into the dictionary for absolute, relocatable, and complex expressions. A diagnostic will be issued if the name has been previously defined (duplicate symbol), if the expression type or length is invalid, or if the symbol in the operand has not been previously defined.
7. If the current logical order file entry represents a CNOP statement, RESCON is called to resolve any conditional alignment. The operand is evaluated and, if valid, the location counter will be aligned to a halfword boundary. If the alignment amount is nonzero, an alignment LOF entry is constructed indicating the number of bits to skip. The alignment entry is inserted in the logical order file preceding the entry for the CNOP statement. The number of generated NOPRs required to satisfy the CNOP is then determined, and the total instruction length (in bits) of the NOPRs is inserted in the LOF entry for the CNOP.
8. If the current logical order file entry represents a machine instruction, a test is made to determine if the location counter is positioned at a halfword boundary. If not, a special entry is made to RESCON to compute the amount of alignment required

and to generate an alignment LOF entry. The alignment entry is inserted in the logical order file preceding the entry for the current machine instruction.

A flag is set in the LOF entry during Phase I if there is a literal in the operand of a statement. If this flag is on, RESLIT is called to scan the literal as if it were a normal DC-statement operand, to prepare a constant item for it, and to enter the literal as an item in the main dictionary.

A test is made to determine if there is a symbol in the name field. If there is, LOCATE is called to assign the current location counter value to the name.

9. If the current logical order file entry represents a CCW statement, RESCON is called to align the current location counter to a doubleword boundary. If the alignment amount is nonzero, an alignment LOF entry is constructed indicating the number of bits to skip. The alignment entry is inserted in the logical order file preceding the entry for the CCW statement.

If the flag bit in the LOF entry is on, indicating a literal in the operand, RESLIT is called. The literal will be scanned (by CSCAN) as if it were a DC-statement operand, a constant item will be prepared for it, and an item for the literal will be entered in the main dictionary.

If there is a symbol in the name field, LOCATE is called to assign the current location counter value to the name.
10. When a logical order file entry representing a MACRO statement is encountered, Phase IIB will step through the LOF entries representing the macro definition until the MEND LOF entry is encountered. Since no processing is required in Phase IIB for a macro definition, the statements are simply bypassed.
11. Other assembly instructions require no special processing and are therefore bypassed.
12. At the end of Phase IIB, the control routine causes the construction of logical order file entries which simulate a CSECT statement (a continuation of the first CSECT) and a LTORG statement. GSM entries are also

constructed to indicate the change of section. The highest value of the location counter for the CSECT is reinstated and the literal pooling routine (POOLIT) is called. All remaining literals that are not address constants are pooled at the end of the first CSECT.

13. Phase IIB now determines whether a designated PSECT exists. If it does, logical order file entries are constructed to simulate a PSECT and a LTORG; GSM entries are constructed to indicate the change of section; the location counter value is set to its highest for the PSECT; and POOLIT is called again, this time with an override switch set that causes the routine to accept address constants when they are encountered in the list of unpooled literals. If no PSECT exists, the override switch is set at the beginning of the phase. This action causes address constants to be pooled with other literals at each LTORG statement.
14. At the end of the phase, the chain of section name items is processed to compute the page usage for the program. Each control section that contains text will have, in the dictionary item, a pointer to its page usage table. To determine the number of pages used by each section, it is necessary to count the bits that have been turned on in the table. The total bit count (total pages) is posted in the section name item for use by Phase III in generating text. A cumulative total of pages for all control sections is computed so Phase III can call VMGET for the required number of pages for the binary text module.

PHASE IIC FUNCTIONAL DESCRIPTION

Phase IIC tabulates the status of PRINT control, LTORG numbers, and USING registers in relation to each control section when a section has been written discontinuously. It also associates the operands of ENTRY statements with the names of control sections in such a way that R-type addressability is established.

An overview of Phase IIC function is shown in Figure 9. The numbered paragraphs in the following description correspond to the numbered boxes in the figure.

Construction of the output module requires Phase III to process each control section contiguously. Phase IIC is required to maintain compatibility with OS/360 definition of PRINT, LTORG, USING, and DROP statements while processing in control section order.

1. By Phase IIC only section names, PRINT, LTORG, USING, DROP, and ENTRY statements remain in the GSM chain. Phase IIC searches the GSM chain and constructs and maintains a working using table. If an ISD is to be produced in Phase IV, an ISD list of using table locations is established. The PRINT, LTORG, USING, DROP, and ENTRY links are removed from the GSM chain as processed.
2. At each section change, the working version of the using table is copied into a permanent location as the current table. Current pointers are updated to address the current control section. The GSM entry for the section name will be followed immediately by an entry pointing to the current using table, which will reflect the status of using registers, PRINT control, and the current LTORG number at the point of continuation.
3. When a LTORG entry is encountered in the GSM chain, the LTORG number in the working using table is updated to the next higher number. The GSM entry for the LTORG is then removed from the GSM chain.
4. If a GSM entry for a PRINT statement is encountered, the new PRINT status is recorded in the working version of the using table. The GSM entry for the PRINT is then removed from the GSM chain.
5. If a GSM entry for an ENTRY statement is encountered, an entry trailer is constructed and linked to the previous entry trailer for the control section. If a trailer has not been previously constructed, the current trailer is linked to the appropriate control section dictionary item. The GSM entry for the ENTRY is then removed from the GSM chain.

After Phase IIB, definitions are available for any symbol that may legitimately appear as an ENTRY operand. The section name within which the ENTRY occurs is also known, since the GSM chain includes section names that Phase IIC records in the current control section address.

If the ENTRY occurs within a named section that is not a DSECT, entry-operand items are constructed in the main dictionary and chained to the item for the named section that is currently in control. This produces definitions that are capable of R-type references. The ENTRY statement may not appear in a DSECT or an unnamed CSECT. ENTRY statements may appear in named common control sections.

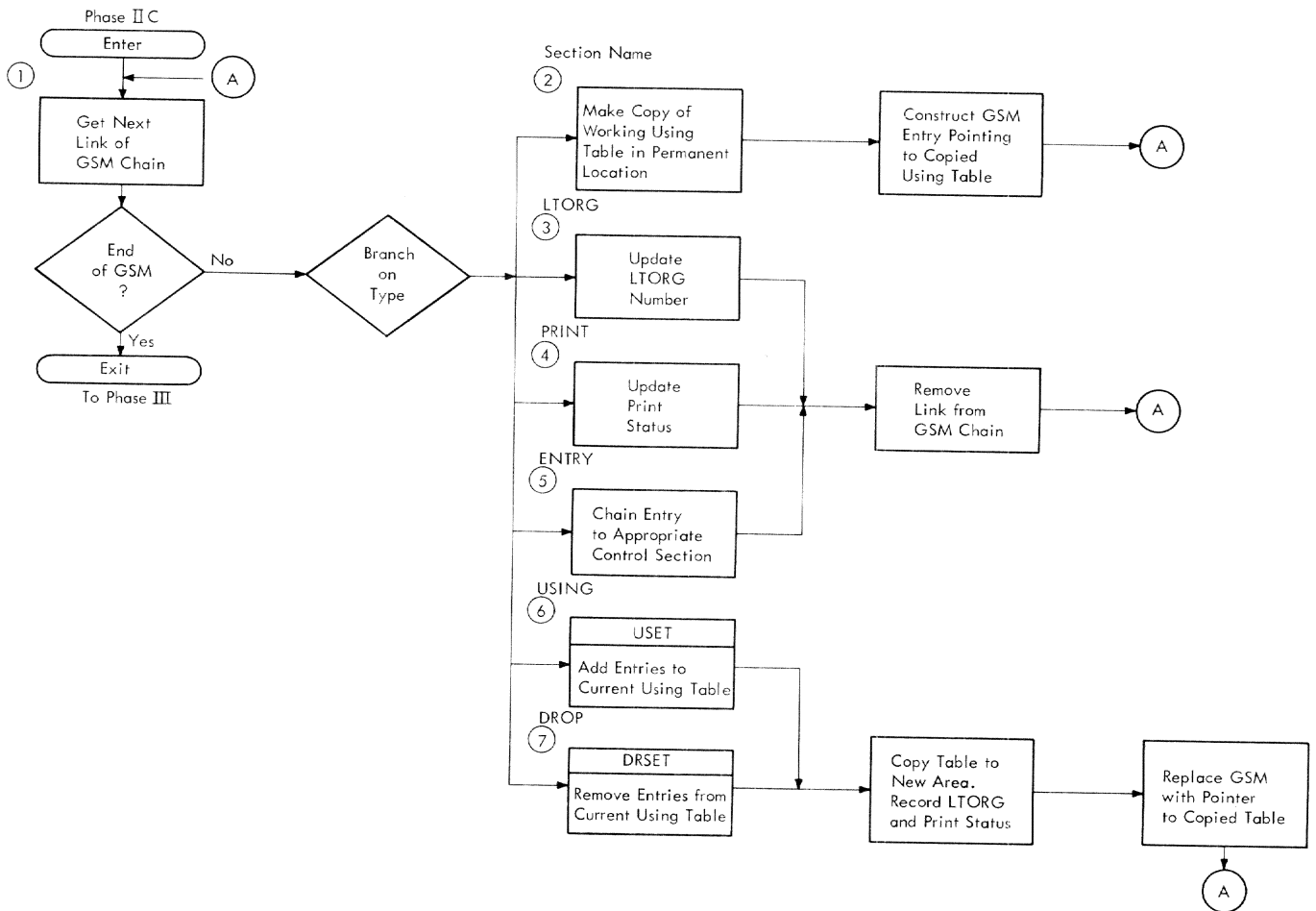


Figure 9. Overview of Phase IIC function

6. The USING table processor, USET, is called by Phase IIC control when the current GSM entry indicates the presence of a USING statement. USET updates the status of the working version of the using table.

USET first resets a series of indicators which it uses to check for duplicate register specifications. It then calls EVAL to evaluate the first operand, which is the base value for the using registers. Absolute and relocatable expressions are acceptable. The relocatable expression may consist of a single external or internal symbol plus or minus any absolute value. Indicators are set to denote whether the table entry is to be in absolute, relocatable, or external format. The base value is set accordingly. Each of the remaining operands is submitted in turn to EVAL. The expression must be absolute, less than 16, and must not duplicate another operand. If legal, the using table entry for the specified register is constructed. USET adds

4096 to the base value for each legal operand after the first until the list of operands is completed. The logical order file entry for the USING statement is completed. Indicators are inserted showing the type of base expression and its value.

The working version of the using table is copied into a permanent location. The GSM entry for the USING is replaced with a base register table locator entry which points to the copied table.

7. The DROP table processor, DRSET, is called by Phase IIC control when the current GSM entry indicates the presence of a DROP statement. DRSET updates the working version of the using table.

DRSET first resets a series of indicators which it uses to check for duplicate register specifications. It then calls EVAL to evaluate the expression for each of the operands. Each expression must be absolute, less than

16, and must not duplicate another operand. If legal, the table entry for the specified register is marked as no longer available as a cover register.

The working version of the using table is copied into a permanent location. The GSM entry for the DROP is replaced with a base register table locator entry which points to the copied table.

PHASE III FUNCTIONAL DESCRIPTION

Phase III controls final processing of all instructions. It organizes the program by control section, produces the necessary binary text and relocation information for the object program, and provides listings of the source and object programs.

An overview of Phase III function is shown in Figure 10. The numbered paragraphs in the following description correspond to the numbered boxes in the figure.

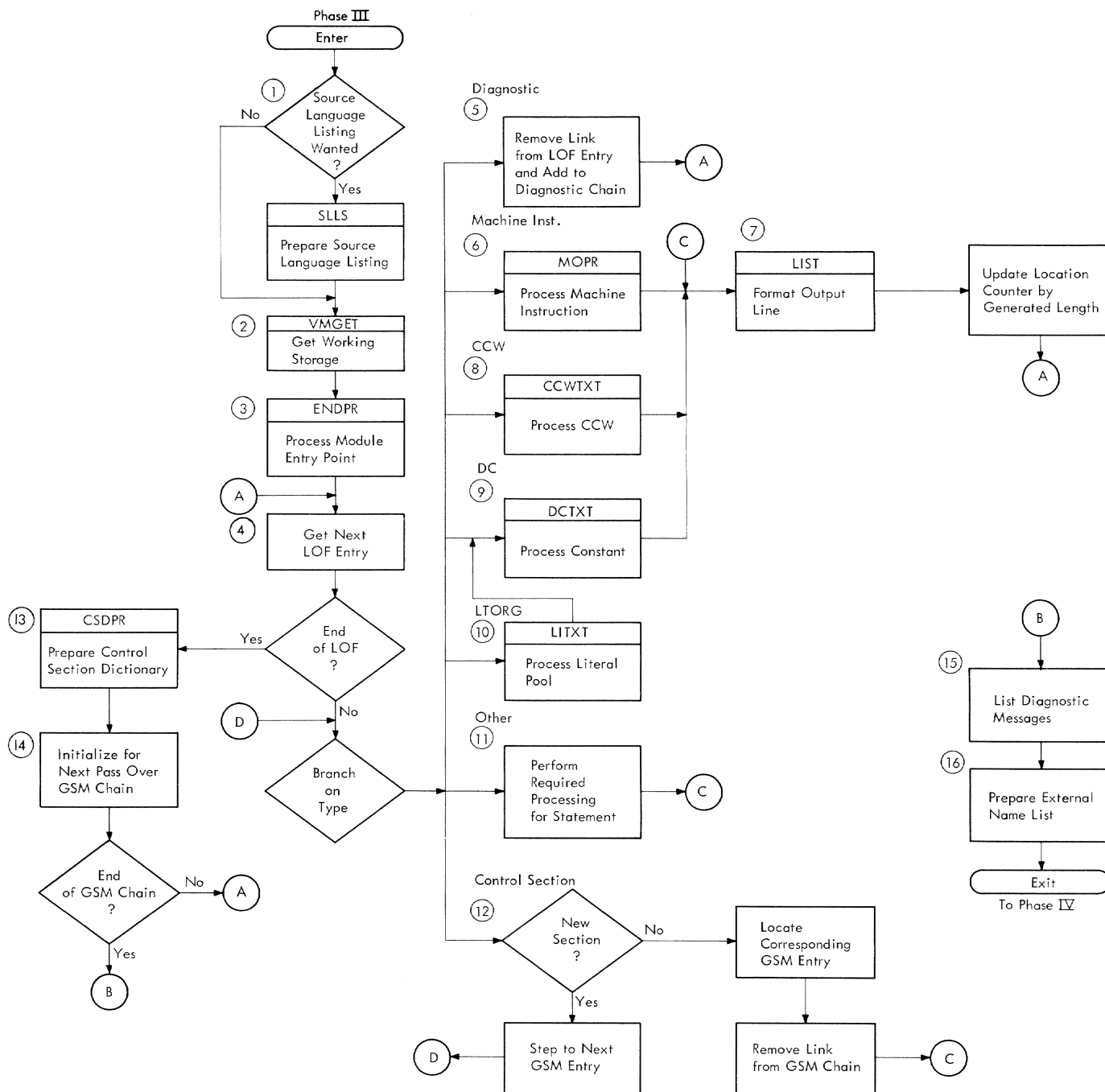


Figure 10. Overview of Phase III function

1. Phase III begins by preparing a source program listing, if one is desired. The source listing processor (SLLS) prepares a listing of the input source lines in their original order and format, with the sequence number assigned to the statement by the line data set facilities. Either the VISAM PUT macro is used to place the edited lines in the list data set, or the GTWRC macro is used to put the listing on SYSOUT, depending on user request and mode. Using the LISTDS operand, the user, in either conversational or nonconversational mode, may direct requested listings to SYSOUT or have them entered in a list data set.
2. The page usage estimated for the output text is calculated, and VMGET is called to procure output working storage. VMGET is called again to procure working storage for the program module dictionary and the external name list.
3. The module entry point processor (ENDPR) is called to construct the module heading and to complete the heading, as far as possible, at this time. ENDPR calls the expression evaluator to evaluate the operand of the END statement and completes the heading according to the type of operand. The length of the module heading is computed, and the location of the first control section dictionary is established.
4. Phase III uses the GSM chain to put the program into order by control section. Within each section, the logical order file controls the order of processing. Each statement represented in the logical order file is processed by an appropriate open or closed subroutine.
5. If a logical order file entry representing a diagnostic is encountered, the entry is delinked from the LOF and added to the diagnostic chain for ease of listing at the end of the phase. The error flag is set with the appropriate code so that the listing routine (LIST) can output the code with the statement to which the diagnostic refers.
6. If a logical order file entry representing a machine instruction statement is encountered, the machine-operation processor (MOPR) is called to evaluate the operand field of the statement and to create corresponding binary output in the text portion of the output program module. The address in the output text which the

instruction is to occupy is calculated prior to entry. The instruction length is determined from the LOF, and the bytes to be occupied by the assembled text are set to zero. The operation code is transferred from the LOF entry to the text. Processing proceeds according to the instruction type: RR, RR with extended M1 field, RR with only one register, RR with immediate value, RX, RX with extended value, RS with explicit R3 field, RS without R3 field, SI with immediate value, SI without immediate value, SS with two length fields, and SS with one length field. The syntax of the operand field for the instruction type is evaluated and checked for validity. As each component field of the instruction is evaluated, the corresponding binary output is placed in the text. When the text has been completed the instruction is checked against the machine operations requirement table to diagnose alignment errors and improper register usage.

Relocatable operands are submitted to the USEVAL subroutine, which reduces the relocatable symbol to a base register and displacement value. The location counter value of relocatable operands, including literals, is obtained by the GETVAL subroutine. Exit is made to Phase III control.

7. The object program listing routine (LIST) is called to format the output line. LIST uses the current logical order file entry and is supplied the location and length of any binary text generated for the statement represented by the LOF entry. With these sources of information LIST can prepare a suitably formatted line for the object program listing. If PRINT control is set to OFF, LIST performs no processing. If ON, LIST prints information relative to the binary text on the left side of the listing and information relative to the source statement on the right. LIST uses the PUT macro in VISAM to place each line in a list data set, or the GTWRC macro if the listing goes immediately to SYSOUT. The line is 132 characters and is preceded by an ASA FORTRAN standard print control character: blank for single space, 0 for double space, and 1 for page eject.

The location counter for the current control section is incremented by the generated length, if any, of the current statement. The address in the output text that the next instruction is to occupy is computed accordingly.

8. If a logical order file entry representing a CCW instruction is encountered, the CCW instruction processor (CCWTEXT) is called to evaluate the operand field of the statement and to create corresponding binary output in the text portion of the output program module. The address in the output text that the CCW is to occupy is calculated prior to entry. Adjustment will have been made to a doubleword boundary. The eight bytes of text are set to zero. The syntax of the four operands is evaluated and checked for validity. As each component field of the instruction is evaluated, the corresponding binary output is placed in the text. The PUTVAL subroutine is called for relocatable data address operands, including literals, to create the necessary relocation dictionary information required to modify the text of a relocatable field. Exit is made to Phase III control.

The LIST routine is next called to format the output line. For a description of its function, refer to item 7.

9. If a logical order file entry representing a DC statement is encountered, the Phase III constant processor (DCTXT) is called to place the binary text for the constant into the output module. The text and relocation values for address constants not previously obtained are resolved during this processing. DCTXT examines the constant item associated with the LOF entry for the DC and establishes a duplication factor for the text. If the constant is an address constant, the ADCON subroutine is called to produce text and relocation information for the constant. If the constant is not an address constant, its value is retrieved from the constant item and moved to the text location. For bit-length constants the text location is bit-oriented.

Movement of data into the text is repeated until the duplication factor is reduced to zero. The LIST routine is called to generate printed output on each duplication when the DATA print option is specified, except for bit-length fields. For bit-length fields, the next LOF entry is tested when all duplications of the current constant are complete. If the next LOF entry indicates a multiple-operand bit-length constant, the bit-oriented text location is maintained at its current updated value so that the next constant may be packed at the next adjacent bit. The entire bit-length

constant is then printed. Multiple operands for non-bit-length constants are processed by successive entries to DCTXT.

After return is made to Phase III control, the location counter for the current control section is incremented by the generated length of the constant. The address in the output text that the next instruction is to occupy is computed accordingly.

10. If a logical order file entry representing a LORG statement is encountered, the literal pooling processor (LITXT) is called to place the binary text for the literals in the given pool into the output module. The values of address constants not previously obtained are resolved during this processing. The logical order file entry points to the head of a chain connecting all literals pooled under the given LORG. For each literal in the pool, LITXT creates an artificial source line for the benefit of the listing. LITXT also constructs an artificial LOF entry for the literal, simulating an entry for a normal DC statement. Having made the literal appear as if it were a normal constant, LITXT calls the DCTXT routine to process the constant. DCTXT will, in turn, call LIST to format the output line. Exit is made to Phase III control where the page usage for the pool is determined and, if necessary, entries are made in the virtual storage page table. The address in the output text that the next instruction is to occupy is computed.
11. Various assembler instructions require little processing in Phase III. The processing required for these instructions is described below.

MACRO/MEND: A flag is set when a logical order file entry representing a MACRO instruction is encountered. This flag causes all statements (except diagnostics) that occur until the MEND LOF is encountered to be recognized as model statements in a macro definition and, as such, to require listing only. The LIST routine is called to list each statement.

USING/DROP: The occurrence of a logical order file entry representing a USING or DROP statement indicates that a new using table is required for references by USEVAL. The address of this table is retrieved, the GSM entry for the USING or DROP is delinked, and the LIST routine is called to list the statement.

Alignment: If an alignment LOF entry is encountered, a test is made to determine if text is required. If so, the text is set to zero, page usage is determined, and the LIST routine is called to format the line. In either case, the location counter will be incremented by the generated length.

ORG: If a LOF entry representing an ORG is encountered, the object program location counter is set to the value of the ORG, and the LIST routine is called to format the line.

DS: If a LOF entry representing a DS is encountered, the increment to the location counter is retrieved from the constant item, and the LIST routine called to format the line. The location counter will be incremented by the length of the instruction.

CXD: If a LOF entry representing a CXD is encountered, a CXD reference item and temporary RLD item are built. The CXD is chained onto the external reference chain, and the CXD-REF flag is set on. A call is then made to the LIST routine to format the line.

DXD: The occurrence of a logical order file representing a DXD instruction requires that a Q reference item be built. The DXD instruction is then treated as a DC statement.

CNOP: If a LOF entry representing a CNOP is encountered, and text is required for the CNOP, page usage will be determined, and the text location updated. The number of NOPR instructions required are generated and placed in the output text. The LIST routine is called to format the line; the location counter is incremented by the generated length, and page usage is determined. If text is not required for the CNOP, the LIST routine is called to format the line.

PRINT: The occurrence of a PRINT LOF entry requires only that the indicator for print status be updated.

SPACE: The occurrence of a SPACE LOF entry requires that the operand of the statement be evaluated, and the value plus one set as a parameter for the LIST routine. If the operand is null, the value is set to 2.

TITLE: The occurrence of a TITLE LOF entry requires that a parameter be set to cause a page to be ejected, and the length and location of the title set for the LIST routine.

EJECT: The occurrence of an EJECT LOF entry requires only that a parameter be set to cause a page to be ejected.

Other: Other assembler instructions require listing only.

12. When a logical order file entry representing a control section is encountered, a test is made to determine if the section is new or is a continuation of the section being processed. If it is a new section, the GSM chain will be stepped through to attempt to locate a GSM entry for the current control section. If an entry is not found, all the processing has been performed for the section, and it is time to build the control section dictionary. If an entry is found, the link is removed from the GSM chain, the logical order file entry pointer is positioned to the corresponding LOF entry, the statement is listed, and processing of the LOF will resume at the entry following the entry for the continuation of the current control section.
13. When the end of the logical order file is reached, or the end of the GSM chain when not at the end of the LOF, the control section dictionary processor (CSDPR) is called to complete the processing of a control section dictionary before Phase III control begins processing a new section. CSDPR is responsible for retrieving all relocation modifiers and reference items in temporary storage and for producing a final output CSD from them. CSDPR initially constructs a section heading in the PMD. The total number of bytes in the text and the relative page number where the text begins are placed in the section heading, the section name is located in the dictionary, and CSDPR begins following the chain of ENTRY names that is attached to the section-name item. Three passes are made over the ENTRY chain: (1) the simply relocatable definitions receive definition items constructed in the control module; (2) absolute definitions receive definition items; and (3) complex definitions receive definition items. Similar processing is performed for the RLD modifiers for the text. Finally, a table is constructed in the module with one entry for each page of virtual memory represented by the text.
14. If any links remain in the GSM chain, the first control section entry that remains in the chain is located, the logical order file pointer is positioned to the corresponding LOF entry,

and processing of the logical order file is resumed. If the GSM chain is exhausted, all processing for control sections has been completed.

15. A diagnostic chain has been constructed by transferring diagnostic logical order file entries from the LOF to the diagnostic chain. The LIST routine is now called to list a message for each entry in the diagnostic chain. Summary messages indicating the number of messages and the highest severity code are then produced.
16. A list of external names is prepared by following the chain of section-name items. This list of names is prepared so that LPC can "STOW" them when disposing of the unit.

PHASE IV FUNCTIONAL DESCRIPTION

Phase IV calls the post processors required to produce the output options selected by the programmer. The post processors produce the symbol table listing, cross-reference listing, PMD listing, ISD, and ISD listing. Certain combinations of these services are available to the programmer. In Phase IV, the option flag for each processor is checked, and the post processor is called if its output is desired.

An overview of Phase IV function is shown in Figure 11. The numbered paragraphs in the following description correspond to the numbered boxes in the figure.

1. If the programmer has selected the option for a cross-reference listing, the cross-reference listing processor (XREF) is called to sort the cross-reference items produced during Phase III and to produce an orderly listing of them. In Phase III, cross-reference items were stacked contiguously in working segment 1. A pass is made over the items to produce the listing. The items are sorted alphabetically by key of dictionary item, with definitions preceding references, and references sorted by ascending value of location counter. The address list produced by the sort controls the order of the printed items. The formatted lines are stacked behind the listing in the listing module.
2. If the programmer has selected the option for a symbol table listing, the symbol table editor (STED) is called by Phase IV control. STED prepares a sorted listing of all symbols contained in the main dictionary, together with their type, length, and value

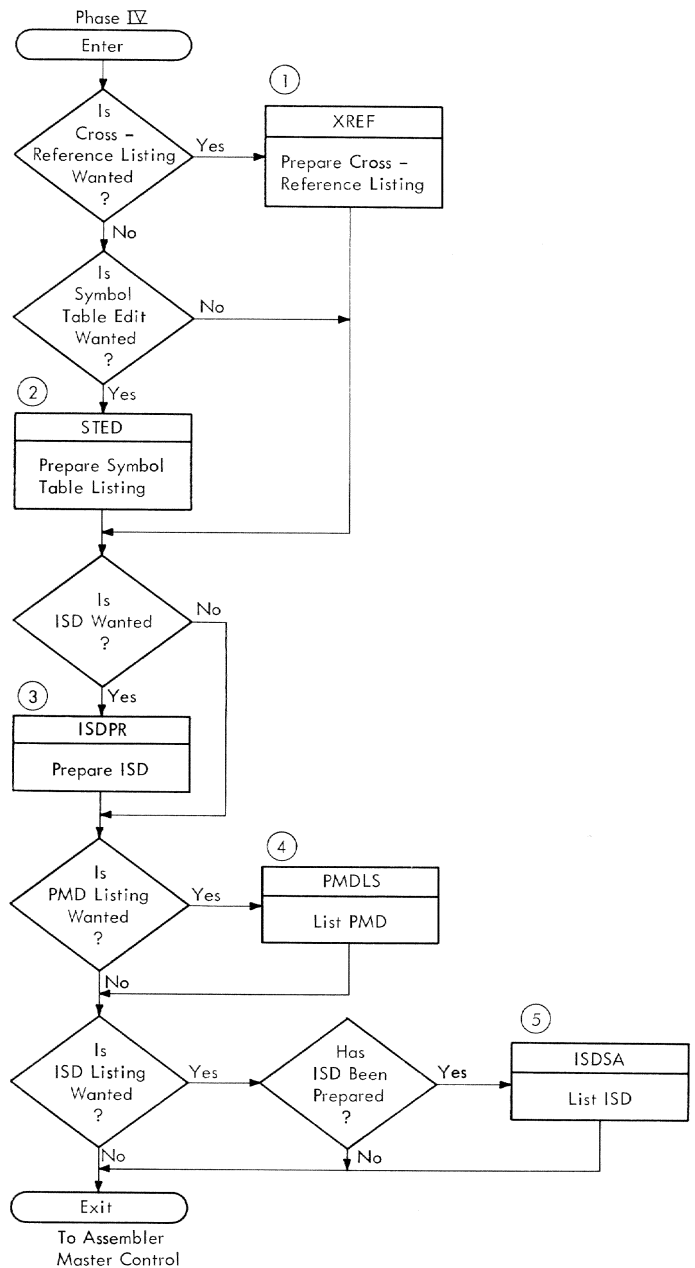


Figure 11. Overview of Phase IV function

attributes. STED follows each link indicated by the main hashing table and stores a sorting key which consists of the address of each item in the main dictionary, except transitive items. STED then sorts the keys into ascending alphameric sequence based on the character value of the symbols in the dictionary. The resulting list is edited for printing, with two columns of symbols appearing on each page.

3. If the programmer has selected the option for an ISD, the ISD processor (ISDPR) is called by Phase IV control to reduce the contents of the permanent dictionary to those items

required by the program control system (PCS), and to format those items conveniently for PCS in a special internal symbol dictionary (ISD).

4. If the programmer has selected the option for a PMD listing, the program module dictionary listing processor (PMDLS) is called by Phase IV control. Information for the listing header lines is secured from the PMD header. The following details, when present, are listed for each control section within the module:

- Section name
- Type of section
- Version identification
- Attributes

- Length of the control section
- Text length
- Relocatable, absolute, and complex definitions for the section
- References
- DXD and CXD references
- Modifiers for complex definitions
- Modifiers for text (internal and external references, Q-CONs, and CXDs)

5. If the programmer has selected the option for an ISD listing, and has also requested an ISD, the ISD list processor (ISDSA) is called by Phase IV control to display the contents of the internal symbol dictionary.

SECTION 3: ASSEMBLER FUNCTION BY INSTRUCTION TYPE

INTRODUCTION

The next three figures show the function of the assembler by type of instruction. The level of nesting is in order from top to bottom and, in general, the sequence of processing is from left to right.

MACHINE INSTRUCTIONS

Figure 12 shows the assembler function for machine instructions.

During Phase I, the REED routine obtains the source statement and a logical order file entry is constructed. If the name field contains a valid symbol, an entry is made in the main dictionary. If the assembly is in conversational mode, the operand field is checked for validity.

Machine instructions generated by macros are processed during Phase IIA, as described below.

During Phase IIB, machine instructions are recognized and the location counter is stepped. If necessary, the location counter value is adjusted to a halfword boundary. If a symbol is present in the name field, it is assigned the value of the current location counter.

Machine instructions are not processed during Phase IIC.

During Phase III, each operand is analyzed for syntactical correctness and checked for validity. The appropriate number of bytes of binary text is generated (by MOPR) and placed in the output text module. The statement is listed on the object program listing.

Machine instructions require no further processing in Phase IV.

MACRO INSTRUCTIONS

The assembler function for macro instructions is shown in Figure 13.

During Phase I, the REED routine obtains the macro reference statement. A logical order file entry and a global-section-macro (GSM) entry are constructed for the statement.

Phase IIA processes the GSM chain and calls the statement analyzer to expand the macro. The routines enclosed by dotted

lines in Figures 12 and 14 are executed for each generated statement. Macro statement generation is accomplished by substituting the character-string values of the current arguments for the corresponding parameters in the definition. The statements generated by macro instructions are created and placed in the assembler virtual storage. If the macro reference is to a library macro, the macro definition must be retrieved from the library, and lines linked together in storage before macro expansion can begin. In Phase IIA, the Phase IIA control module replaces the LPC in determining the order and origin of the statements.

In Phase IIB, the only processing required for a macro reference is to assign the current location counter value to the symbol in the name field (if one exists). Generated statements are processed as machine or assembler source statements during the remainder of the assembler.

In Phase III, the only processing required is to list the statement on the object program listing.

Macro instructions require no further processing in Phase IV.

ASSEMBLER INSTRUCTIONS

The assembler function for assembler instructions is shown in Figure 14.

During Phase I, the REED routine obtains the source statement, and a logical order file entry is constructed. The entire statement is checked for syntactical correctness. If the instruction is one of the following, a global-section-macro (GSM) entry is made: control section statement (CSECT, PSECT, DSECT, COM, START), GBLx, SETx, USING, DROP, ENTRY, PRINT, or LTORG.

During Phase IIA, all GBL declarations, global SET instructions, and section name changes are reprocessed in order that macro-generated statements have proper values for global variables and the assembler variable symbol &SYSECT.

During Phase IIB, only certain types of assembler instructions are processed. EQUATE is called for an EQU; ORIGIN is called for an ORG; POOLIT is called for a LTORG (after first assigning the current location counter value to the name if the line is named); RESCON is called for a CCW followed by RESLIT, if the data address

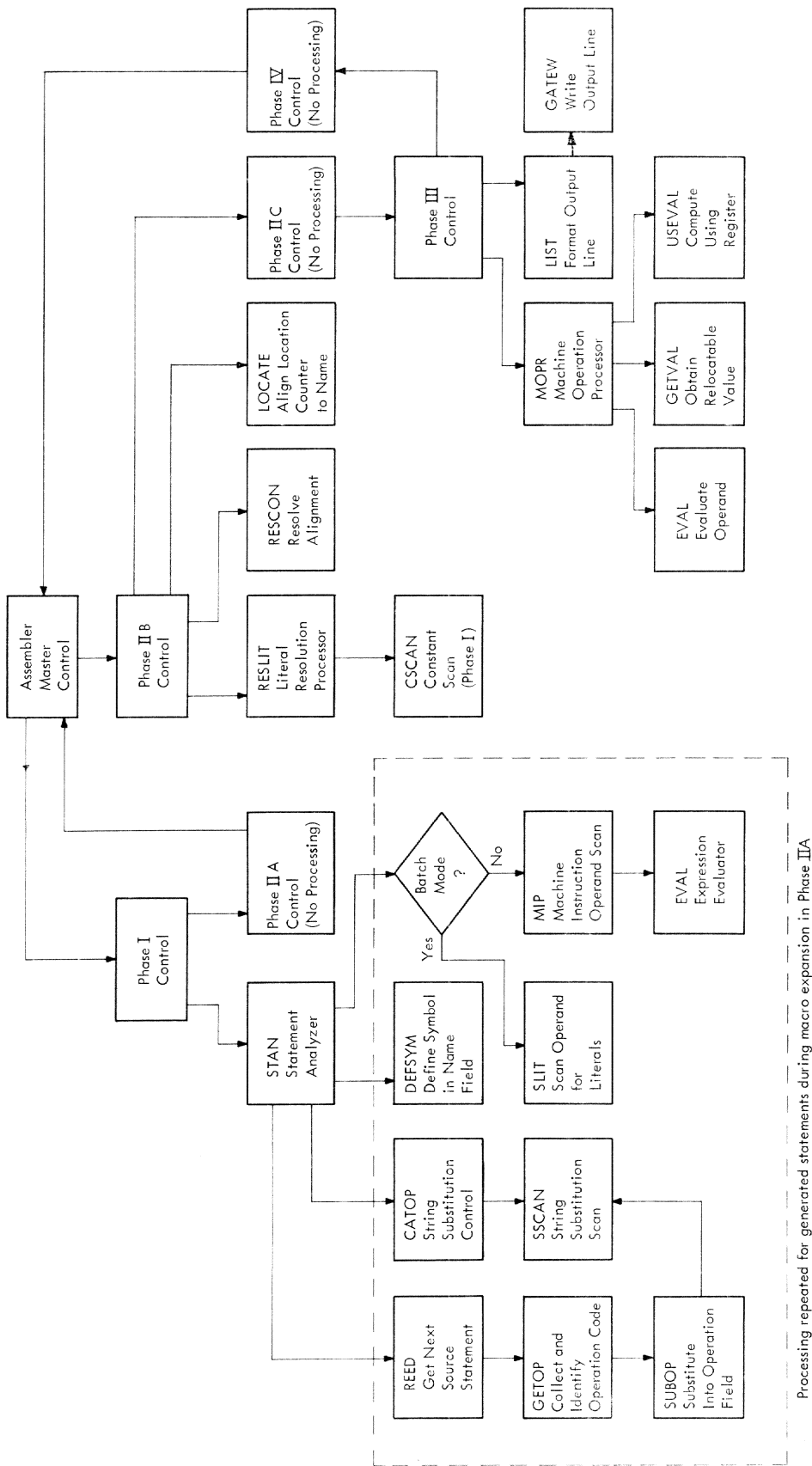


Figure 12. Assembler function for machine instructions

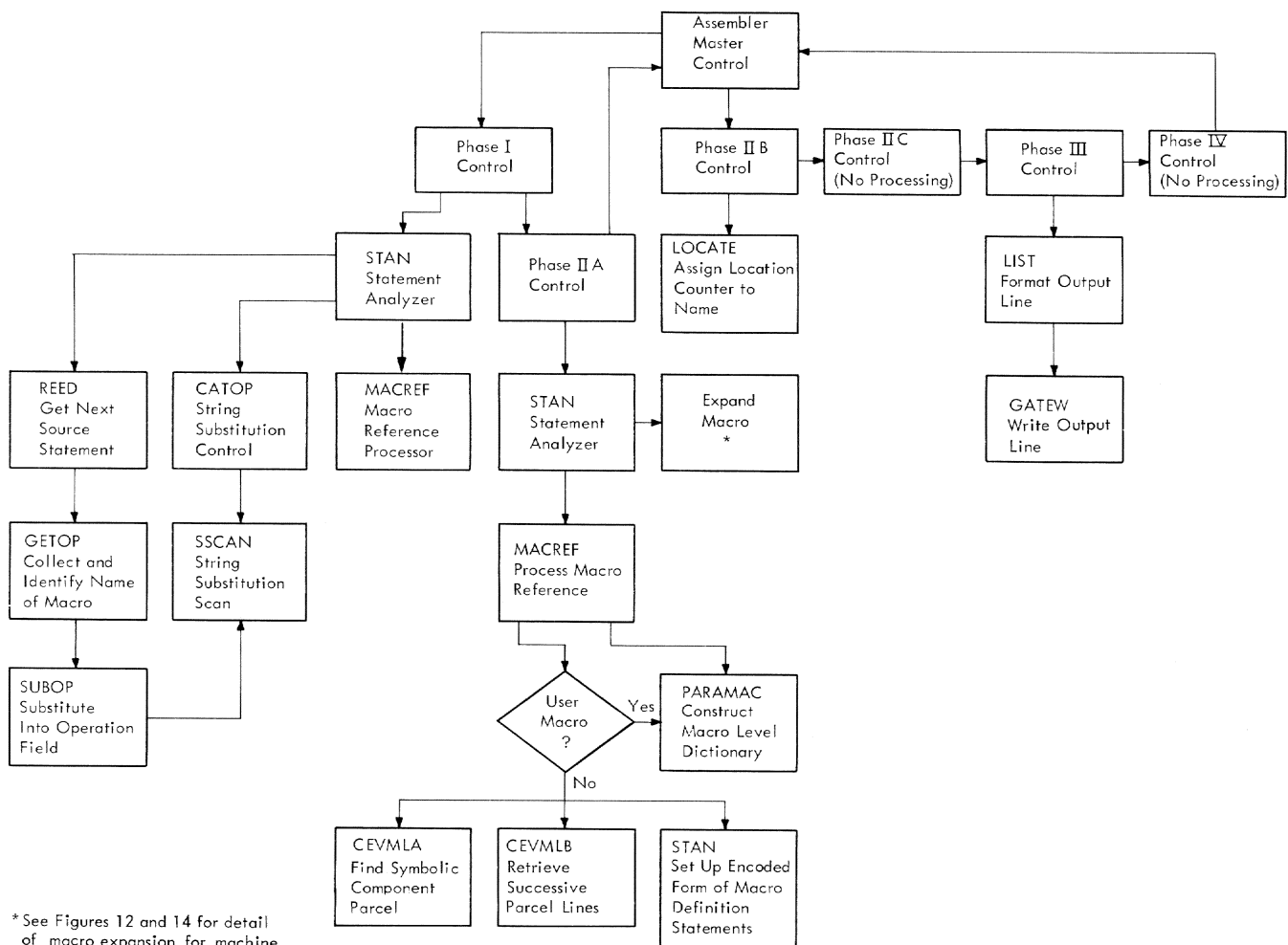


Figure 13. Assembler function for macro instructions

contains a literal, and LOCATE if a symbol exists in the name field. If a constant is being processed (DC or DS), CSCAN will be called to build a constant item if an item was not made in Phase I; RESCON will be called to align the constant and LOCATE will be called to assign the current location counter value to a symbol in the name field. Other assembler instructions are bypassed in Phase IIB.

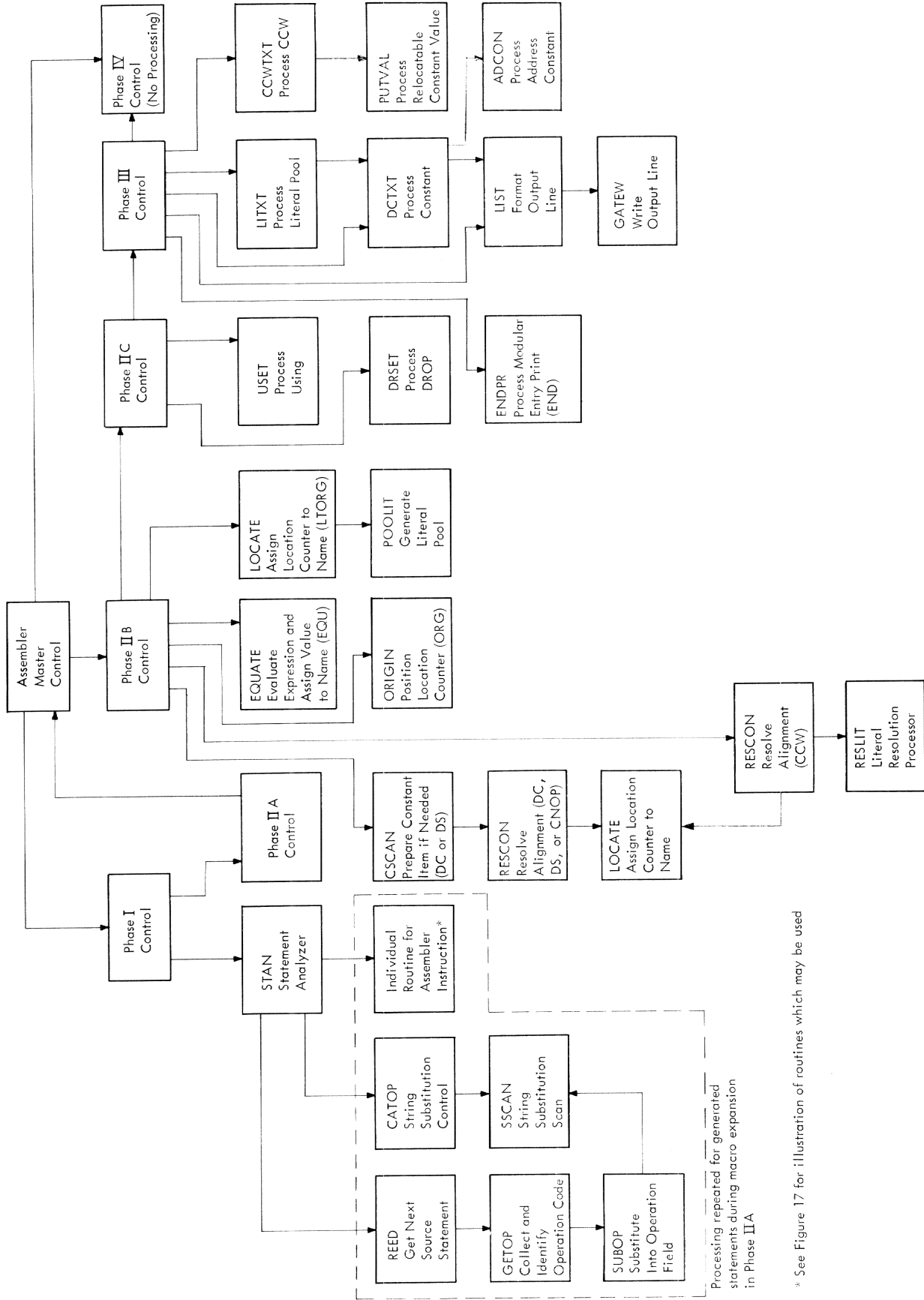
Only the following assembler instructions are processed in Phase IIC: section name (CSECT, DSECT, PSECT, COM, START), LTORG, PRINT, ENTRY, USING, or DROP. USET will be called for a USING statement and DRSET for a DROP statement; each of these statements, as well as a section name statement, result in a new using table being constructed.

The module entry point processor is called first in Phase III to process the END statement. The GSM chain controls the order of processing in Phase III. Because

the output binary text and object listing must be in order by control section, a section name GSM entry will be either accepted as a continuation of the current control section and the processing of the logical order file resumed at the section continuation, or it will be bypassed if it is not the current section and will be processed in a subsequent pass over the GSM chain.

Other types of LOF entries that receive special processing in Phase III Control are MACRO, MEND, EQU, USING, DROP, ORG, DS, CXD, DXD, CNOP, PRINT, SPACE, TITLE, and EJECT. LOF entries that result in special routines being called are: CCW, for which CCWTEXT is called to generate a line for each literal in the pool and which, in turn, calls DCTXT to construct the binary text for each literal. LOF entries for all statements are processed by the LIST routine that will output the object listing.

Assembler instructions require no further processing in Phase IV.



* See Figure 17 for illustration of routines which may be used

Figure 14. Assembler function for assembler instructions

SECTION 4: ASSEMBLER MASTER CONTROL

INTRODUCTION

For purposes of program maintenance, the assembler master control provides a centrally located point of interface between the language processor control (LPC) and the assembler. There are three entry points to the assembler from the LPC: initiation, continuation, early-end.

Figure 15 shows the control flow from phase to phase. The arrows of Figure 15 are one-directional and show the true flow of control in the assembler. As illustrated, the assembler may be considered three subroutines:

- Phases I and IIA, executed upon assembler initiation entry.
- Phases IIB, IIC, III, and IV, executed upon assembler continuation entry.
- Early-end processing performed entirely in the assembler control module.

Phase calling conditions are specified in Tables 1 and 2 followed by the detailed description of the assembler control.

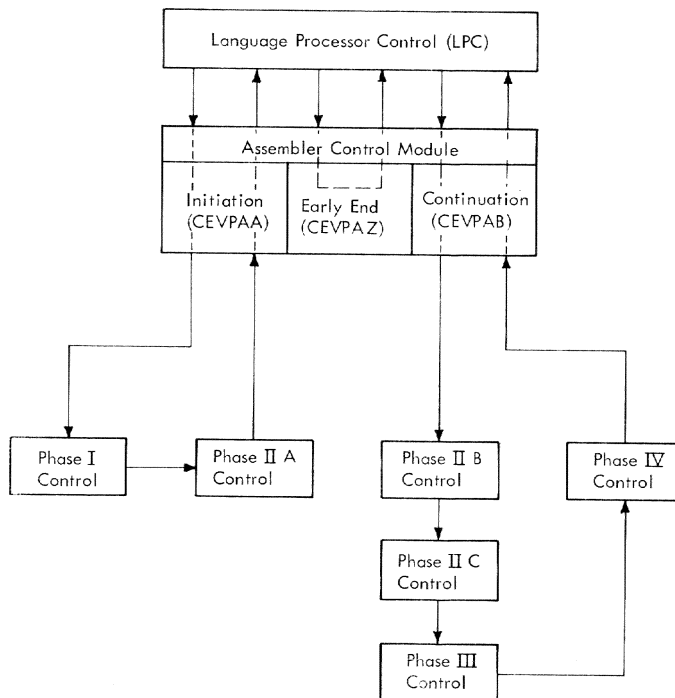


Figure 15. LPC calls and assembler phase control flow

Table 1. LPC call to AC

Routine: Language Processor Control			
Routine	Purpose	Called Routines	Calling Conditions
Language Processor Control (LPC)	Provides communication between terminal and Assembler.	AC -- CEVAC	Always called for: Phase I initialization Phase IIB continuation Abnormal assembler termination

Table 2. Assembler control decision table (part 1 of 2)

Routine: Assembler Control Level: 0			
Routine	Purpose	Called Routines	Calling Conditions
AC (CEVAC)	Provides a centrally located interface with the language processor control.	PHASE I -- CEVPA	Assembler initialization.
		PHASE IIB -- CEVPC	Assembler continuation.
		VMFREE -- CEVFM	At normal end, to free unused virtual storage.
		VMCLEAN -- CEVCU	Early end: to clear up storage obtained from GETMAIN. Normal end: to reset VMTABLE.

Table 2. Assembler control decision table (part 2 of 2)

Routine: Level: 1			
PHASE I	Reads and performs initial processing of source program.	PHASE IIA -- CEVPB (transfer of control)	Completion of Phase I processing.
PHASE IIA (CEVPB)	Expands macro instructions and obtains library macro definitions.	AC -- CEVAC (transfer of control)	Completion of macro expansions.
PHASE IIB (CEVPC)	Aligns all statements to the required boundary, computes page usage for each control section, resolves literal references, pools and assigns location counter values to literals and resolves symbol definitions.	PHASE IIC -- CEVPD (transfer of control)	Phase IIB completion.
PHASE IIC (CEVPD)	Tabulates the status of PRINT control, LTOrg numbers, and USING registers in relation to each control section.	PHASE III -- CEVPE (transfer of control)	Phase IIC completion.
PHASE III (CEVPE)	Controls the final processing of all instructions; organizes the program by control section, produces text and relocatable information, and provides listings.	PHASE IV -- CEVPF (transfer of control)	Phase III completion.
PHASE IV (CEVPF)	Calls the post processor modules to produce the output options selected by the user.	AC -- CEVAC (transfer of control)	Completion of an assembly.

AC -- Assembler Control (CEVAC)

This routine provides the interfaces necessary to implement the initiation, continuation, and early-end entries to the assembler. (See Chart AA.)

Entry Points: CEVPAA, CEVPAB, CEVPAZ

Calling Sequence: A description of the calling sequence for each entry point follows.

Initiation: The initial entry is a CALL from LPC with register 1 pointing to a parameter list of the following format:

Word	Content
1	Address of a character string containing the module name; the assembler uses only those characters preceding the first period in the string, or the first eight characters, whichever is shorter.

2 Address of a one-byte indicator, which is zero for batch mode, nonzero for conversational mode.

3 Address of an eight-byte table of options; each byte contains values of 'Y' if the option is to be selected, or 'N' if it is not. The options represented by the table are as follows:

Byte	Content
1	Produce ISD
2	Produce source listing
3	Produce object listing
4	Produce cross-reference listing
5	Produce edit symbol table listing
6	Produce PMD listing

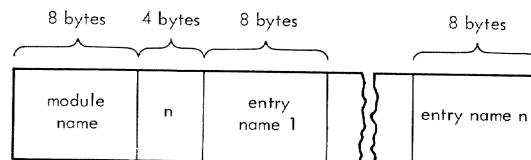
- 7 Produce ISD listing
- 8 Listings to a list data set (rather than SYSOUT)
- 4 Address of the DCB for the list data set, if specified.
- 5 Address of the number of user macro libraries. The number should be six or less; if greater than six, only the first six will be used by the macro service routines.
- 6 Not used.
- 7 Address of the DCB for the source component of the system macro library.
- 8 Address of the DCB for the index component of the system macro library.
- 9 Address of one-byte indicator. If zero, word 10 represents a reading from the system time facility, which is to be used as a version identification for the module; if nonzero, word 10 is a user-supplied character string.
- 10 Address of an eight-byte version identification string containing a system time reading or a user-supplied character string, as indicated by word 9.
- 11 Address of the DCB for the source component of the first user macro library.
- 12 Address of the DCB for the index component of the first user macro library.
- 13-22 Addresses of the DCBs for the source and index components of the second and following user macro libraries. The number of address constant pairs in words 11-22 must equal the count to which word 5 points. If there are no user libraries, words 11-22 will not be used.

Note: The user libraries are searched prior to the system library with the heirarchy of search proceeding backward through the list from the nth to the first library.

Continuation: The continuation entry is a CALL from LPC with register 1 pointing

to a parameter list of the following format:

<u>Word</u>	<u>Content</u>
1	Address of a one-byte indicator, set by the assembler to nonzero if records have been PUT into the listing data set and zero if no records have been PUT into the listing data set.
2	Address of a one-word field set by the assembler at exit to the length of the PMD in bytes, expressed as a binary integer.
3	Address of a one-word field set by the assembler at exit to the starting virtual storage address of the PMD.
4	Address of a one-word field set by the assembler at exit to the length of the assembled program text, in bytes (to the nearest page multiple), expressed as a binary integer.
5	Address of a one-word field set by the assembler at exit to the starting virtual storage address of the assembled program text.
6	Address of a one-word field set by the assembler at exit to the length of the ISD in bytes, expressed as a binary integer.
7	Address of a one-word field set by the assembler at exit to the starting virtual storage address of the ISD.
8	Address of a one-word field set by the assembler at exit to the starting virtual storage address of a list of external names (entry points) defined by the assembled module. The list has the following format:



n - The number of entry names (excluding module name)

Early-end: The early-end entry is a CALL from LPC with register 1 pointing to the address of a one-byte indicator, as described under "Continuation," word 1.

Routines Called: None.

- Macro Instructions Issued

OPEN (VISAM)
CLOSE (VISAM)
SAVE
RETURN

Exits:

- PHASE I (CEVPA) - Initiation
- PHASE IIB (CEVPC) - Continuation
- Return to LPC - Early-end or completion of assembly.

OPERATION: The operations performed vary for each entry point.

Initiation: Calling registers are saved, and the assembler's permanent registers are loaded. A test is made whether requested listings go back to the user on SYSOUT or into a list data set. If a listing data set is to be created, a VISAM index key is initialized and the data set is opened; otherwise, a SYSOUT switch is set and the VISAM data set initiation is bypassed. Control is then transferred to Phase I. Control returns at the end of Phase IIA,

calling registers are restored, and an exit is made.

Continuation: Calling registers are saved, and the assembler's permanent registers are restored. Control transfers to Phase IIB control, and the remaining phases of the assembler are executed. Control returns at the end of Phase IV. All output parameters required by LPC are filled in, and a return code is calculated based on the diagnostic severity code. Unused pages acquired for the ISD and PMD are released with FREEMAIN; all working storage is freed; and the listing and macro library data sets, if any exist, are closed. Unless restart procedures are in effect, calling registers are restored and control returns to LPC. The early-end entry is made and portions of the continuation routine are used for restart in response to an altered line. When this is the case, control returns once again to Phase I, as if initiation had been invoked.

Early-end: Calling registers are saved and the assembler's permanent registers are restored. All storage acquired for the ISD and PMD is freed. A return code for early-end is set, and control passes to that portion of the continuation routine where the remainder of working storage is freed. Control returns to LPC with the early-end code.

SECTION 5: PHASE I

INTRODUCTION

Phase I functions are performed by a collection of specialized routines to process each source language statement in order, and to produce the main dictionary, the macro-name dictionary, the global-section-macro chain, and a partially encoded version of the program (logical order file), from which Phase III produces the output program module.

As shown in Figure 16, the statement analyzer governs the main flow of control. It is employed in Phase I to process the original source statements and again in Phase IIA to process the source statements generated by the expansion of macro instructions. Thus, the main body of Phase I code may be reexecuted during Phase IIA.

Phase I communicates with external routines in four instances:

- When REED (obtain next source statement) requests the next line from the language processor control GETLINE subroutine.
- When a macro instruction for which there is no user macro definition is encountered in source statements (see GETOP).
- When COPY statements are encountered.
- When conversational diagnostic messages are produced.

The recursive call made by CSCAN upon EVAL to process DC and DS statements should be noted. Initially, DC/DS calls CSCAN, which calls EVAL, which calls CSCAN, etc. In Figure 16, the arrow from EVAL to CSCAN serves a double function:

1. It shows the recursive calls to and from CSCAN and EVAL in processing DC and DS statements.
2. It shows the nonrecursive call by EVAL on CSCAN when processing machine instructions or an assembler instruction other than DC or DS.

Although not shown in Figure 16, all processors marked with an asterisk (*) should show arrow communication with the diagnostic message processor (DIAG).

Table 3 specifies the conditions under which the Phase I routines are called.

ROUTINES

PHASE I -- Phase I Control (CEVPA)

This routine accepts the operating parameters from the language processor control (LPC), reads and performs the initial processing of the source program, and transfers control to PHASE IIA. (See Chart AB.)

Entry Point: CEVPAX

Calling Sequence: L R15,ACEVPA
BR R15

Input Parameters: R1 - location of input parameter pointer list

Routines Called: DLPM, STAN, VMGET, VMFREE

Exit: To PHASE IIA (CEVPB)

OPERATION: The pointer to the list of pointers to the input parameters is contained in register R1. The input parameters are listed in the description of the assembler control module (AC).

Upon receiving control, VMGET is called to acquire two areas of virtual storage for Phase I's own working storage requirements and one area for the source statements. These areas are assigned to each user's virtual storage. Initial and default values and beginning addresses for variable storage are inserted into the static portion of working storage. Static working storage is also modified as a result of the operating parameters transmitted by the language processor control. The date and time are obtained from the REDTIM macro, and DLPM is called to build dictionary items for &SYSDATE and &SYSTIME.

Having established the source program data set as the current input source, control is transferred to STAN for the program to be processed. When the end of the program has been reached, control is returned to Phase I, which, in turn, transfers control to Phase IIA.

STAN -- Statement Analyzer (CEVST)

This routine is a control program, which uses a collection of other routines to process each source language statement in order, and to produce the main dictionary and a partially encoded version of the program, logical order file, from which the output program module is produced during Phase III. (See Chart AC.)

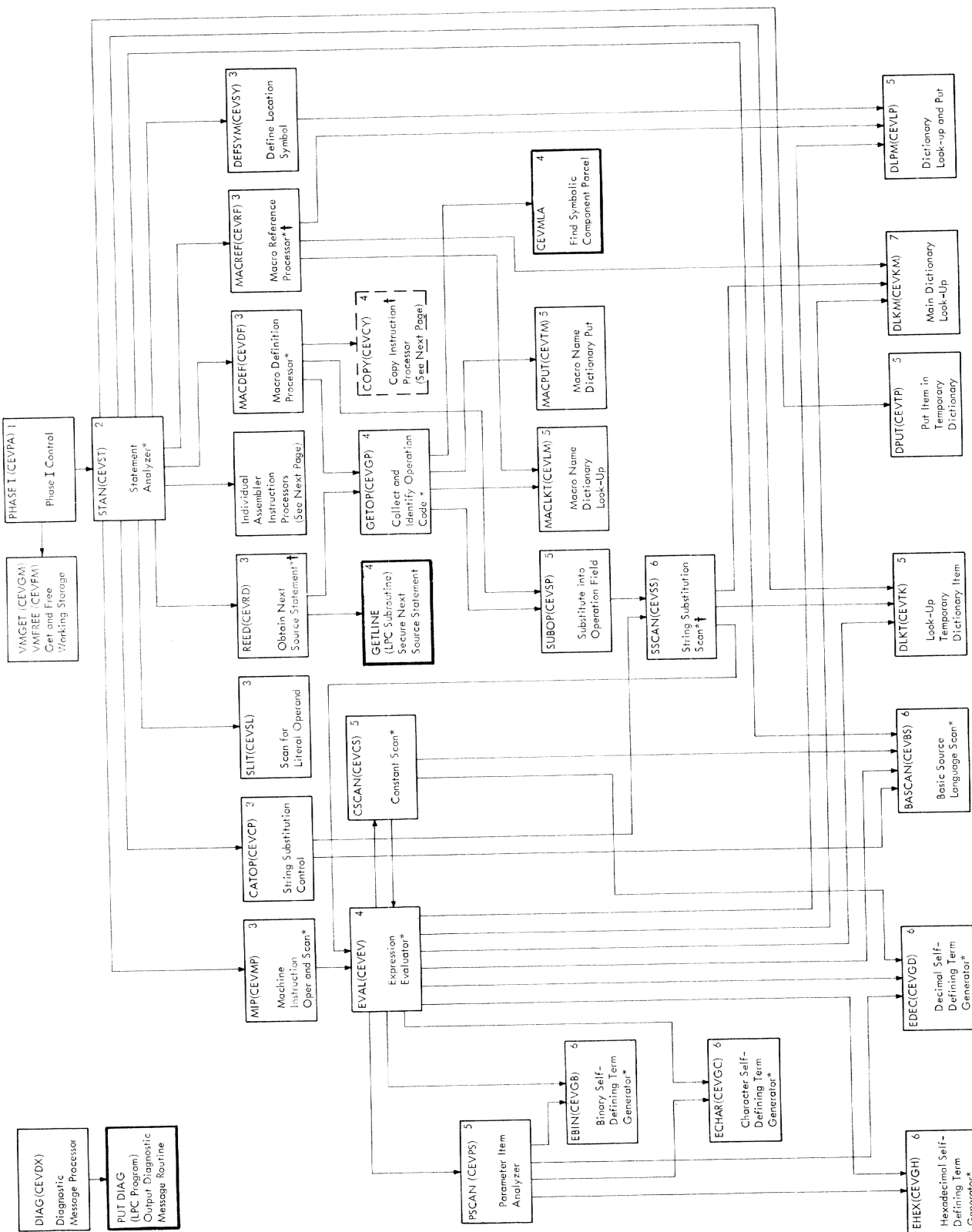


Figure 16. Phase I routine relationships (part 1 of 2)

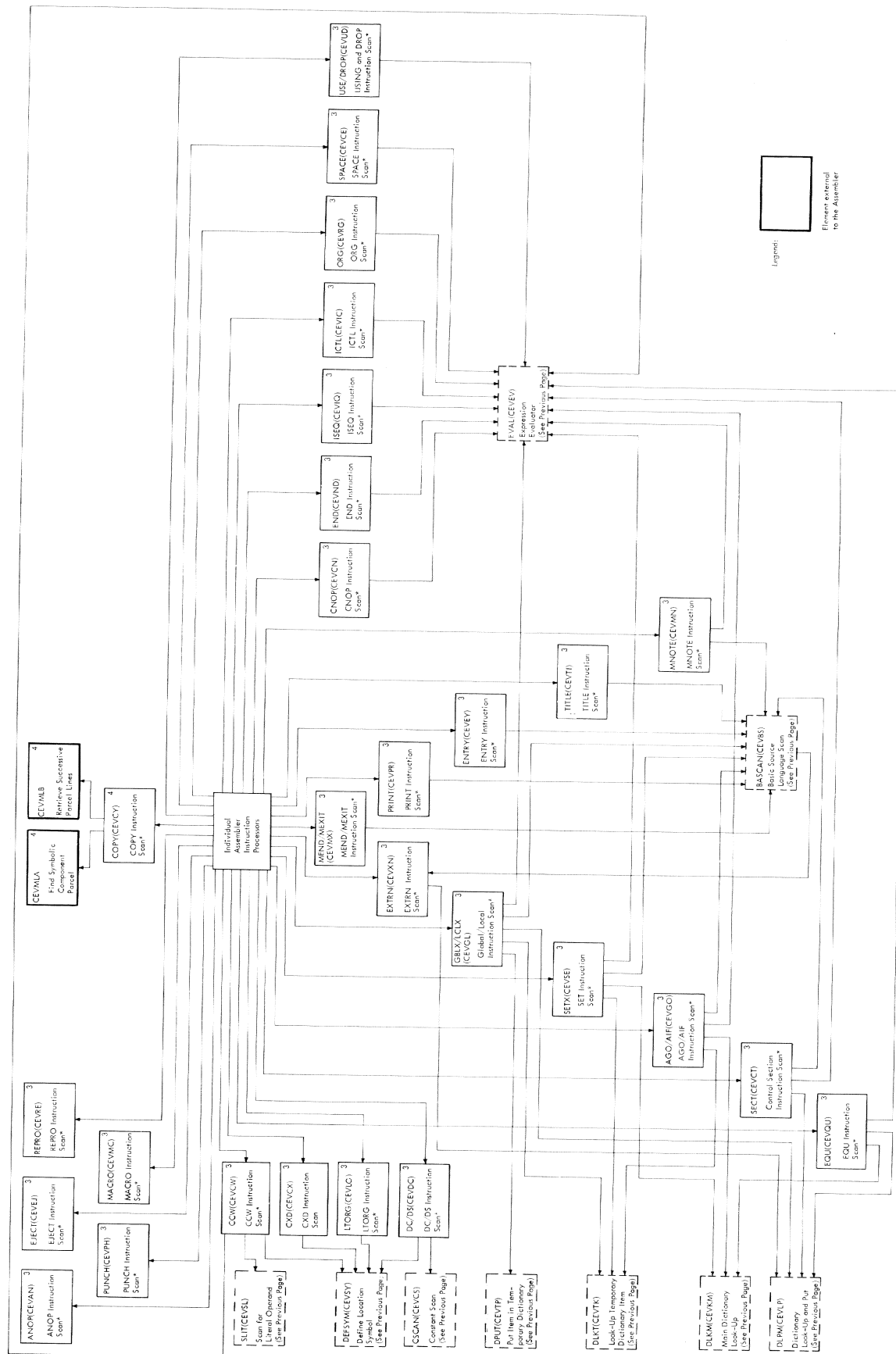


Figure 16. Phase I routine relationships (part 2 of 2)

Table 3. Phase I decision table (part 1 of 14)

Routine: Phase I Control Level: 1			
Routine	Purpose	Called Routines	Calling Conditions
PHASE I (CEVPA)	Accepts parameters from LPC; reads and performs initial processing of source program; and transfers control to Phase IIA.	DLPM -- CEVLP	Always called.
		STAN -- CEVST	Always called.
		VMGET -- CEVGM	When necessary to acquire working storage.
		VMFREE -- CEVFM	To release storage.
Routine: Level: 2			
STAN (CEVST)	Processes original source statements during Phase I, and macro-generated instructions during Phase IIA.	AGO/AIF -- CEVGO	AGO or AIF instruction encountered.
		ANOP -- CEVAN	ANOP instruction encountered.
		BASCAN -- CEVBS	Sequence symbol encountered in name field.
		CATOP -- CEVCP	Always called.
		CCW -- CEVCW	CCW instruction encountered.
		COPY -- CEVCY	COPY instruction encountered.
		MIP -- CEVMP	Machine instruction encoun- tered while in conversational mode.
		CNOP -- CEVCN	CNOP instruction encountered.
		CXD -- CEVCX	CXD instruction encountered.
		SECT -- CEVCT	Control section statement encountered.
		DC/DS -- CEVDC	DC or DS statement encountered.
		DEFSYM -- CEVSY	Machine instruction encountered.
		DIAG -- CEVDX	D102 Undefined sequence symbol D106 Missing MEND statement D110 Missing END statement
		DLPM -- CEVLP	Sequence symbol encountered in name field, or CSECT implied.
		USE/DROP -- CEVUD	USING or DROP statement encountered.
		EJECT -- CEVEJ	EJECT statement encountered.
		END -- CEVND	END statement encountered.
ENTRY -- CEVEY	ENTRY statement encountered.		

Table 3. Phase I decision table (part 2 of 14)

Routine: Level: 2 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
		EQU -- CEVQU	EQU statement encountered.
		EXTRN -- CEVXN	EXTRN statement encountered.
		GBLX/LCLX -- CEVGL	GBLA, GBLB, GBLC, LCLA, LCLB, or LCLC statement encountered.
		ICTL -- CEVIC	ICTL statement encountered.
		ISEQ -- CEVIQ	ISEQ statement encountered.
		LTORG -- CEVLG	LTORG statement encountered.
		MACDEF -- CEVDF	Macro prototype of model statement encountered.
		MACREF -- CEVRF	Macro reference statement encountered.
		MACRO -- CEVMC	MACRO statement encountered.
		MEND/MEXIT -- CEVMX	MEND or MEXIT statement encountered.
		MNOTE -- CEVMN	MNOTE statement encountered.
		ORG -- CEVRG	ORG statement encountered.
		PRINT -- CEVPR	PRINT statement encountered.
		PUNCH -- CEVPH	PUNCH statement encountered.
		REED -- CEVRD	Processing of current statement completed.
		REPRO -- CEVRE	REPRO statement encountered.
		SETX -- CEVSE	SETA, SETB, or SETC statement encountered.
		SLIT -- CEVSL	Machine instruction encountered while in batch mode.
		SPACE -- CEVCE	SPACE statement encountered.
		TITLE -- CEVTI	TITLE statement encountered.
		DLKT -- CEVTK	Sequence symbol in name field of an instruction within a macro.
		DPUT -- CEVTP	Sequence symbol in name field of an instruction within a macro which has not been previously included in temporary dictionary.

Table 3. Phase I decision table (part 3 of 14)

Routine: Level: 3			
Routine	Purpose	Called Routines	Calling Conditions
AGO/AIF (CEVGO)	Processes AGO and AIF instructions.	EVAL -- CEVEV	First character of an AIF operand is a left parenthesis
		BASCAN -- CEVBS	First character of an AGO operand is a period, or character following right parenthesis in an AIF operand is a period.
		DLKM -- CEVKM	Sequence symbol encountered in operand field of AGO or AIF instruction, which occurs in user-level code.
		DLKT -- CEVTK	Sequence symbol encountered in operand field of AGO or AIF instruction, which occurs in a macro.
		DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in field D95 Sequence symbol missing in operand field D115 Invalid operand field
ANOP (CEVAN)	Scans the name field of ANOP instructions for a sequence symbol.	DIAG -- CEVDX	D96 No sequence symbol in name field
CATOP (CEVCP)	Controls the type and amount of parameter variable symbol substitution applied to the current source statement.	BASCAN -- CEVBS	Always called unless an error return is made from CEVSS.
		SSCAN -- CEVSS	Always called unless the statement is GBL- or LCL-.
CCW (CEVCW)	Examines the CCW instruction for valid operand fields and correct format.	DEFSYM -- CEVSY	Always called.
		SLIT -- CEVSL	Always called in batch mode.
		EVAL -- CEVEV	Always called in conversational mode.
		DIAG -- CEVDX	D1 Improperly delimited field D6 Invalid expression type for field D14 Required operand missing
CNOP (CEVCN)	Scans CNOP instructions for valid expressions and correct format.	EVAL -- CEVEV	Always called to examine operand in conversational mode.
		DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in name field D6 Invalid expression type for field D7 Invalid expression value for field D14 Required operand missing

Table 3. Phase I decision table (part 4 of 14)

Routine: Level: 3 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
CXD (CEVCX)	SCANS CXD instruction and validates name field.	DEFSYM -- CEVSY	Always called.
DC/DS (CEVDC)	Scans DC and DS instructions.	DEFSYM -- CEVSY	Always called.
		CSCAN -- CEVCS	Always called to examine operand.
		DIAG -- CEVDX	D131 Invalid or missing name field in DXD statement
DEFSYM (CEVSY)	Constructs a main dictionary item for all statements with a symbol in the name field, except when the symbol is encountered a second time.	DLPM -- CEVLP	Symbol in name field.
		DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in field D4 Symbol in name field previously defined
EJECT (CEVEJ)	Scans the EJECT instruction for correct format.	DIAG -- CEVDX	D2 Invalid symbol in name field.
END (CEVND)	Scans the END instruction for correct format and validity.	EVAL -- CEVEV	Always called to examine operand in conversational mode.
		DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in field D11 Invalid operand on END statement. D113 END statement encountered within a macro
ENTRY (CEVEY)	Scans the ENTRY instruction for correct format and validity.	DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in name field D89 Entry point declared in control section without SYSTEM attribute D91 Entry point not CZ- or CHB- in privileged CSECT D115 Invalid operand field
		BASCAN -- CEVBS	Always called to examine operand.
EQU (CEVQU)	Scans EQU instructions and evaluates and completes the definition of those expressions which can be resolved.	DLKM -- CEVKM	Symbol encountered in name field
		EVAL -- CEVEV	Always called to examine operand.
		DLPM -- CEVLP	Symbol encountered in name field which has not been previously defined in main dictionary.

Table 3. Phase I decision table (part 5 of 14)

Routine: Level: 3 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
		DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in name field D4 Duplicate symbol D6 Invalid expression type for field D7 Invalid expression value for field D14 Required operand missing D17 Required name field invalid or missing D115 Invalid operand field
EXTRN (CEVXN)	Scans EXTRN instructions for correct format and validity.	BASCAN -- CEVBS	Always called to examine operand.
		DLPM -- CEVLP	Symbol encountered in operand field.
		DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in name field. D19 Symbol in operand field previously defined. D115 Invalid operand field
GBLX/LCLX (CEVGL)	Scans global and local instructions for correct format and validity.	BASCAN -- CEVBS	Always called to examine operand.
		DLPM -- CEVLP	Variable symbol encountered in operand field, when instruction occurs in user-level code.
		DLKT -- CEVTK	Variable symbol encountered in operand field, instruction occurs in a macro.
		DPUT -- CEVTP	Variable symbol encountered in operand field which has not been defined in temporary dictionary.
		EVAL -- CEVEV	Subscripted variable symbol encountered in operand field.
		DIAG -- CEVDX	D1 Improperly delimited field D4 Duplicate symbol D6 Invalid expression type for field D27 Nonblank name field D97 Missing subscript D98 Dimension exceeds 255 D115 Invalid operand field

Table 3. Phase I decision table (part 6 of 14)

Routine: Level: 3 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
ICTL (CEVIC)	Scans ICTL instructions for correct format and validity.	EVAL -- CEVEV	Always called to examine operand
		DIAG -- CEVDX	D1 Improperly delimited field
			D2 Invalid symbol in name field
			D6 Invalid expression type for field
			D7 Invalid expression value for field
D14 Required operand missing			
D51 Statement occurred illegally			
ISEQ (CEVIQ)	Scans ISEQ instructions for correct format and validity.	EVAL -- CEVEV	Always called to examine operand.
		DIAG -- CEVDX	D1 Improperly delimited field
			D2 Invalid symbol in name field
			D6 Invalid expression type for field
D7 Invalid expression value for field			
LTORG (CEVLG)	Scans LTOrg instructions for correct format	DEFSYM -- CEVSY	Always called.
MACDEF (CEVDF)	Enters the name of macros in the macro name dictionary and controls the processing of macro definitions.	COPY -- CEVCY	COPY statement encountered.
		CATOP -- CEVCP	COPY statement encountered.
		DIAG -- CEVDX	D86 Macro name redefines machine operation.
			D87 Invalid operation in macro definition
		MACLKT -- CEVLM	Macro prototype encountered.
		MACPUT -- CEVTM	Macro name not in macro dictionary.
		SUBOP -- CEVSP	Macro prototype encountered.
GETOP -- CEVGP	END used as macro name.		
MACREF (CEVRF)	Controls processing of macro references.	DLKM -- CEVKM	Symbol in name field of macro instruction.
		DLPM -- CEVLP	Symbol in name field of macro instruction not previously defined.
		MACLKT -- CEVLM	Always called during Phase IIA.
		VMGET -- CEVGM	To acquire working storage.

Table 3. Phase I decision table (part 7 of 14)

Routine: Level: 3 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
		CEVMLA (external)	Macro definition not in storage.
		CEVMLB (external)	Macro found in library index by CEVMLA.
		PARAMAC--CEVPM	Always called during Phase IIA.
		DIAG -- CEVDX	D22 Error in macro library retrieval D59 Error in macro definition
MACRO (CEVMC)	Scans MACRO instructions for correct format	DIAG -- CEVDX	D27 Nonblank name field
MEND/MEXIT (CEVMX)	Processes MEND and MEXIT instructions.	BASCAN -- CEVBS	Called to examine MEND name field when in Phase I and the MEND is not a generated statement.
		DIAG -- CEVDX	D2 Invalid symbol in name field D114 Instruction invalid outside macro
MIP (CEVMP)	Scans machine instruction operands for valid operand fields and correct format.	EVAL -- CEVEV	Always called to examine operand.
		DIAG -- CEVDX	D1 Improperly delimited field D6 Invalid expression type for field D7 Invalid expression value for field D10 Attempted store into literal. D14 Required operand missing
MNOTE (CEVMN)	Scans the operand of MNOTE instructions for correct format and validity.	BASCAN -- CEVBS	Statement not previously determined to be invalid.
		EVAL -- CEVEV	Operand does not begin with a quote.
		DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in field D6 Invalid expression type for field D115 Invalid operand field
ORG (CEVRG)	Scans ORG instructions for correct format and validity.	EVAL -- CEVEV	Always called in conversational mode.
		DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in name field D6 Invalid expression type for field

Table 3. Phase I decision table (part 8 of 14)

Routine: Level: 3 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
PRINT (CEVPR)	Scans the PRINT instruction for correct format and validity.	BASCAN -- CEVBS	Always called.
		DIAG -- CEVDX	D1 Improperly delimited field
			D2 Invalid symbol in name field
			D21 Invalid operand for PRINT statement
D118 Inconsistent operand			
PUNCH (CEVPH)	Analyzes the PUNCH instruction.	DIAG -- CEVDX	Always called.
			D23 PUNCH statement produces listing only.
REED (CEVRD)	Read source input.	LPC Get Line -- CFADB1	Phase I processing of current source line completed.
		GETOP -- CEVGP	Current statement other than a macro prototype or macro call.
		DIAG -- CEVDX	D3 Card out of sequence
			D121 Too many continuation lines
D135 Continuation cards have non-blank characters before continue column, characters ignored			
		VMGET -- CEVGM	Get storage for source lines.
REPRO (CEVRE)	Analyzes the REPRO instruction.	DIAG -- CEVDX	Always called.
			D24 REPRO statement produces listing only.
SECT (CEVCT)	Checks the validity of the control section instructions (COM, START, CSECT, DSECT, PSECT).	DLPM -- CEVLP	Symbol or blanks encountered in name field.
		EVAL -- CEVEV	START instruction encountered.
		DIAG -- CEVDX	D1 Improperly delimited field
D2 Invalid symbol in name field			
D4 Duplicate symbol			
D6 Invalid expression type for field			
D7 Invalid expression value for field			
D15 Incompatible control section statements			
D16 Invalid control section attribute name			
D17 Required name field invalid or missing			
D56 Entry point name same as module name			
D89 Entry point declared in control section without system attribute			

Table 3. Phase I decision table (part 9 of 14)

Routine: Level: 3 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
			D91 Entry point not CZ- or CHB- in privileged system CSECT D112 START not first control section statement D118 Inconsistent operand.
		BASCAN -- CEVBS	A COM, PSECT, or CSECT instruction encountered.
SETX (CEVSE)	Scans SETA, SETB, and SETC instructions for correct format and validity.	DLKT -- CEVTK	Macro level greater than zero.
		EVAL -- CEVEV	a) Symbol in name field is subscripted. b) Instruction is a SETB or SETA.
		DLKM -- CEVKM	a) Macro level is zero. b) A type attribute occurs in the operand of a SETC instruction.
		BASCAN -- CEVBS	SETC instruction being processed.
		DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in name field D6 Invalid expression type for field D7 Invalid expression value for field D84 Invalid subscript value D85 SET statement incompatible with definition D93 More than 8 characters in string D97 Missing subscript D115 Invalid operand field
SLIT (CEVSL)	Scan for literals.	None	
SPACE (CEVCE)	Scans the SPACE instruction for correct format and validity.	EVAL -- CEVEV	Always called in conversational mode.
		DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in name field D6 Invalid expression type for field
TITLE (CEVTI)	Scans the TITLE instruction for correct format and validity.	BASCAN -- CEVBS	Always called to examine operand.
		DIAG -- CEVDX	D2 Invalid symbol in name field. D27 Nonblank TITLE name field, and not the first TITLE statement

Table 3. Phase I decision table (part 10 of 14)

Routine: Level: 3 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
			D48 Truncated value D109 TITLE name field exceeds 4 characters D115 Invalid operand
USE/DROP (CEVUD)	Scans USING and DROP instructions for correct format and validity.	EVAL -- CEVEV	Always called in conversational mode.
		DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in field D6 Invalid expression type for field D7 Invalid expression value for field D12 Duplicate use of register D14 Required operand missing D120 Attempted explicit register specification
Routine: Level: 4			
COPY (CEVCY)	Copy lines from library.	CEVMLA	Library is available and operand is a valid symbol.
		CEVMLB	Code to be copied is found in library.
		DIAG -- CEVDX	D14 Required operand missing D26 Cannot be found on copy library D28 Copied statements contained COPY statement D115 Invalid operand field
		VMGET -- CEVGM	To get working storage.
EVAL (CEVEV)	Evaluates an arithmetic or logical expression designated by the calling module and returns with the value and type of expression.	BASCAN -- CEVBS	Always called.
		DIAG -- CEVDX	D5 Undefined symbol D6 Invalid expression type for field D8 Invalid attribute D9 Multiple literals in statement D68 Improperly formed logical expression D69 Symbol not previously defined D74 Unbalanced parentheses in expression D75 Consecutive operators in expression D76 Consecutive terms in expression D78 Excessive parentheses in expression D79 Excessive terms in expression

Table 3. Phase I decision table (part 11 of 14)

Routine: Level: 4 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
			D81 Multiplication or division of relocatable terms D82 Attribute unavailable outside macros D84 Invalid subscript value D88 Invalid symbolic parameter D90 Undefined variable symbol D97 Missing subscript D105 Value of expression causes overflow D115 Invalid operand field
		DLKM -- CEVKM	Symbol is encountered.
		DLPM -- CEVLP	Symbol is encountered.
		CSCAN -- CEVCS	Literal is encountered.
		DLKT -- CEVTK	Variable symbol encountered inside a macro.
		EDEC -- CEVGD	Decimal self-defining term encountered.
		EHEX -- CEVGH	Hexadecimal self-defining term encountered.
		EBIN -- CEVGB	Binary self-defining term encountered.
		ECHAR -- CEVGC	Character self-defining term encountered.
		PSCAN -- CEVPS	Parameter symbol encountered inside a subscript.
GETOP (CEVGP)	Isolates and identifies the operation code mnemonic of current source statement.	SUBOP -- CEVSP	Always called if current statement is not commentary.
		MACLKT -- CEVLM	Operation code not found in operation code table, or if in table, the operation code has been redefined as a macro.
		MACPUT -- CEVTM	Operation code found in the macro name library index.
		DIAG -- CEVDX	D43 Undefined mnemonic operation
		CEVMLA	Definition for user-level macro instruction not provided by user, a library is available, and library not previously searched for the macro name.

Table 3. Phase I decision table (part 12 of 14)

Routine: Level: 5			
Routine	Purpose	Called Routines	Calling Conditions
CSCAN (CEVCS)	Collects and analyzes each subfield of the data definition and produces a constant item for each operand examined.	EVAL -- CEVEV	Nonnumeric duplication factor, length modifier, scale modifier, and/or exponent modifier encountered.
		EDEC -- CEVGD	Numeric encountered in a subfield.
		BASCAN -- CEVBS	Always called.
		DIAG -- CEVDX	D45 Multiple operands in literal D46 Zero duplication factor in literal D48 Truncated value D49 Invalid hexadecimal constant D50 Invalid binary constant D53 FP characteristic out of bounds D54 All precision lost during scaling D57 Improper operand for V-type adcon D58 Improper operand for R-type adcon. D60 Invalid type subfield D61 Invalid length modifier D62 Scale modifier not permitted D63 Exponent modifier not permitted D64 Exponent modifier out of range D65 Scale modifier out of range D66 Multiple constants not permitted D67 Data omitted from DC operand D115 Invalid operand field D116 Invalid decimal constant D132 Improper operand for Q-type adcon
DLKT (CEVTK)	Locates a specified symbol in the temporary dictionary for the current macro level.	None	
DLPM (CEVLP)	Searches for a given symbol in the main dictionary, and creates a skeletal entry for the main dictionary if the symbol is not found.	None	
DPUT (CEVTP)	Creates a skeletal item for the specified symbol in the temporary dictionary, for the current macro level.	None	
MACLKT (CEVLM)	Searches the macro name dictionary for a given name.	None	

Table 3. Phase I decision table (part 13 of 14)

Routine: Level: 5 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
MACPUT (CEVTM)	Inserts a skeletal macro name dictionary item in the main dictionary, and the macro's hash number in the macro name hash table.	None	
PSCAN (CEVPS)	Analyzes parameter arguments.	EBIN -- CEVGB	Binary self-defining term encountered.
		ECHAR -- CEVGC	Character self-defining term encountered.
		EDEC -- CEVGD	Decimal self-defining term encountered.
		EHEX -- CEVGH	Hexadecimal self-defining term encountered.
SUBOP (CEVSP)	Extract operation code from source line.	SSCAN -- CEVSS	& found in op code.
Routine: Level: 6			
BASCAN (CEVBS)	Scans the portion of the source language statement specified by the callers and identifies the syntactic components of that field.	DIAG -- CEVDX	D1 Improperly delimited field D2 Invalid symbol in field D31 Symbol exceeds 8 characters in length D55 End of statement occurred before processing was completed D71 Invalid binary self-defining term D103 Invalid character in statement
EBIN (CEVGB)	Converts a binary self-defining term into its binary equivalent.	DIAG -- CEVDX	D71 Invalid binary self-defining term
ECHAR (CEVGC)	Converts a character self-defining term into its binary equivalent.	DIAG -- CEVDX	D83 Invalid character self-defining term
EDEC (CEVGD)	Converts a decimal self-defining term into its binary equivalent.	DIAG -- CEVDX	D72 Invalid decimal self-defining term
EHEX (CEVGH)	Converts a hexadecimal self-defining term into its binary equivalent.	DIAG -- CEVDX	D70 Invalid hexadecimal self-defining term.
SSCAN (CEVSS)	Perform string substitution.	EVAL -- CEVEV	Subscript encountered.
		DLKT -- CEVTK	Parameter of local variable symbol encountered.
		DLKM -- CEVKM	Global variable symbol encountered.

Table 3. Phase I decision table (part 14 of 14)

Routine: Level: 6 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
		DIAG -- CEVDX	D7 Invalid expression value for field D8 Invalid attribute D84 Invalid subscript value D88 Invalid symbolic parameter D94 Invalid substring notation D97 Missing subscript D107 More than 255 characters in string D108 Substring notation inconsistent with substring length
		VMGET -- CEVGM	To get working storage.
Routine: Level: 7			
DLKM (CEVKM)	Locates a specified symbol in the main dictionary.	None	
DIAG (CEVDX)	Process diagnostic messages.	LPC Get Line -- CFADC1	Always called in conversational mode.

The statement analyzer is employed in Phase I to process the original source statements and again in Phase IIA to process the source statements generated by the expansion of macro instructions.

Entry Points: CEVST, CEVST1

Calling Sequences: INVOKE ASTAN
INVOKE AST090

Routines Called:

AGO/AIF	EJECT	MNOTE
ANOP	END	ORG
BASCAN	ENTRY	PRINT
CATOP	EQU	PUNCH
CCW	EXTRN	REED
CNOP	GBLX/LCLX	REPRO
COPY	ICTL	SECT
CXD	ISEQ	SETX
DC/DS	LTORG	SLIT
DEFSYM	MACDEF	SPACE
DIAG	MACREF	TITLE
DLKT	MACRO	USE/DROP
DLPM	MEND/MEXIT	
DPUT	MIP	

Normal Mode: In the normal mode, source lines are obtained and processed to produce some change in the information compiled by the assembler to further the production of an object program. Depending upon the type of source line, the processing may result in the growth of the logical order file of the program, additional definition entries in the dictionaries, changes in status, the production of diagnostic messages or some combination of these effects.

A skeletal logical order file entry is made for each new statement encountered. Subsequently, each routine that acquires relevant knowledge appropriately updates the entry.

All statements pass through a subroutine, GETOP, whose purpose is to obtain and classify the operation code.

Statements within macro definitions are selected for separate handling, which results in making the macro conveniently available for later reference. All other statements are directed to the CATOP routine. It has the responsibility to perform all required variable symbol and parameter substitution in the name and operand fields. Upon exiting from CATOP, macro instructions and the assembler instructions are split off for individual handling while machine instructions are grouped by type.

Exits: Phase 1. Same exit made for normal completion and error condition.
Phase 2. Return to MACREF.

OPERATION: This routine has two modes of operation; normal and bypass.

Bypass Mode: The bypass mode is initiated by the processing of an AGO or true AIF command whose transfer point is a sequence symbol which is as yet undefined. In this mode, source lines are merely bypassed until a line containing the desired sequence symbol is encountered, at which time normal processing is resumed.

While processing in bypass mode, sequence symbols occurring in the name field are always defined by the construction of a local dictionary item, unless the line is within a macro definition. Otherwise, bypassed statements are not processed in any way except that END statements and, during macro expansion, MEND statements are recognized to prevent an incorrect branch statement from overrunning the source program.

Error Checks:

- Missing MEND statement.
- Undefined sequence symbol.
- Missing END statement.

REED -- Obtain Next Source Statement (CEVRD)

This routine provides the interface with the language processor control (LPC), to obtain source lines. It concatenates continuation lines to provide STAN with a continuous statement, performs sequence checking, and switches the source of input statements between LPC, macro definitions, and COPY library statements. The altered line processing of corrections to the source program is also performed by this module. (See Chart AD.)

Entry Point: CEVRD

Calling Sequence:

INVOKE	AREED
...	end-of-file return
...	normal return 4(R14)

Routines Called:

- Internal - DIAG, GETOP, VMGET
- External - CFADC1 (Get next line entry point of LPC)
- Macro Instructions - CALL

Exit: Normal
End-of-file

OPERATION: During Phase I this routine obtains source lines directly from LPC. During Phase IIA the principal source of

input is previously processed macro definitions.

Regardless of its origin, a source line may be in either keyboard or card image format and a source statement may comprise multiple source lines, through the statement continuation capabilities.

When satisfying a COPY statement or macro definition request, the entire library element is copied line by line into the assembler's working storage by the appropriate subroutine. Then, requests for source lines are processed from the stack of saved lines until the stack is exhausted, whereupon the input source status reverts to that in which the COPY statement or macro instruction was encountered. A macro definition may contain a COPY statement, thus requiring a push down stack.

In servicing a demand for the next source statement, if a continued source line is encountered, all portions of the statement are combined into a single continuous line which is constructed in assembler working storage. Normally, this reconstruction is a blind process, since the begin, end, and continue columns are clearly defined for both card and keyboard formats. However, when the macro definition switch (MDS) is set to 1, declaring that the current statement is a macro prototype, or when analysis of the operation code indicates a macro instruction statement, logic is applied to determine whether the source lines are in the alternate card or keyboard statement format. Thus, line continuations are handled solely in the raw input routine, and the remainder of the processor sees only continuous, simply scannable statements.

This module is also responsible for performing and commenting diagnostically upon failures in the sequence check demanded by the prevailing ISEQ requirements.

Language Processor Control Input: The LPC contains an entry point for the assembler's use when the next source line is required. The assembler calls the LPC with a line number (initially zero) and LPC responds by returning the source message corresponding to the next line number in sequence. The line number is expected to occupy one word and be in packed decimal format with seven integer places and a sign digit that is positive at all times. The information supplied to the assembler:

- The location of the first byte of the source message.
- The length of the message in bytes.

- A format indicator (keyboard or card image).
- The line number of the message.

Alternate Modes of Input: Input is obtained from three sources other than its LPC interface: COPY (and MACRO) library lines, macro definition statements, and previously processed source statements at the user level (in response to a backward AGO statement).

Library lines are the unedited source lines of macro definitions or COPY elements. When such lines are required by the assembler (in response to a COPY instruction or a reference to a library macro definition), COPY or MACREF initiates appropriate input/output activity and copies all the lines of the library element into working storage. The lines are sequentially chained by a control word preceding the line. In COPY mode, the chain is followed and source lines are procured from working storage until the end of the chain is reached. The previous input source mode is then reinstated from a push-down list.

Macro definition statements are statements which have previously passed through STAN, which have logical-order-file entries, and which have had continuation lines removed. In macro definition mode the logical-order-file is followed to procure each statement, until it reaches a MEND statement, whereupon it reinstates the previous input mode and exits with an end-of-input indication.

For a backward AGO statement at the user level, outside of macro definitions, the source statement control chain is followed as for COPY mode, except that the statements have had continuation lines removed, as in macro definition mode; hence, continuation line processing is bypassed.

Source Program Correction Facility: The assembler provides the conversational user with the ability to correct or delete the last source statement presented to the processor without incurring restart of the entire assembly.

This capability is provided by this module by recording the internal status of the assembler as each source statement is completed. Thus, at any time prior to commencing the processing of the next statement, the effect of the current statement can be erased by replacing the current status information with the previous status, and by detaching from linkage chains any dictionary items constructed since the previous status was recorded.

Determination of Changes: Three line numbers are maintained to assist in determining the effect of a source program change upon the assembly. SLINE is the number of the last line received from LPC. PLINE is the number of the last line of the last statement assembled. This statement is represented by the "current status" and is not final. ALINE is the number of the last line of the next-to-last statement assembled. This statement is represented by the "previous status" and is final.

If LPC returns the assembler's call for the next line with an "altered line" code, the changed line having the lowest number is examined. If its line number is greater than SLINE, a condition which can occur on restarts, the change has no effect on the assembly, and the line following SLINE is again requested.

If the changed line is not greater than SLINE but is greater than PLINE, the change has occurred during the accumulation of continuation lines for a statement which has not yet been seen by STAN. The lines presently accumulated are erased, SLINE is set to the value of PLINE, and the assembly continues by requesting the line following PLINE.

If the changed line is not greater than PLINE but is greater than ALINE, the previously assembled statement is invalid. The current status indicators are replaced by the previous ones, hash linkages in the main hash table which point to storage used since the previous status was recorded are restored, and the previously assembled statement is effectively erased. SLINE and PLINE are set to the value of ALINE in order to prevent a second attempt to erase before a new statement has been received. The assembly continues by requesting the line following ALINE.

GETOP -- Collect and Identify Operation Code (CEVGP)

This routine isolates and identifies the operation code mnemonic of the current source statement. (See Chart AE.)

Entry Point: CEVGP

Calling Sequence: INVOKE AGETOP

Routines Called:

- Internal - DIAG MACPUT
 MACLKT SUBOP
- External - CEVMLA

Exit: Normal

OPERATION: Lines that start with * or .* and contain only a name, or are entirely blank in the statement area, are recognized as commentary.

Any parameter or variable symbol substitution indicated in the operation code is satisfied immediately. The resultant character string is subjected to look-up in the following order:

- Operation code table.
- Macro name dictionary for a match to a macro definition item.
- Macro-library search (via CEVMLA).

The failure of the preceding steps results in diagnosing the mnemonic code as undefined and treating the statement as commentary, unless the statement is a macro model statement or the assembler is in bypass mode. In these cases the mnemonic is ignored.

If a macro name is found in the macro name dictionary, the location of the item is placed in the logical order file entry. If the item is found after searching the macro library, the library position information is saved for the macro reference processor routine (MACREF) for later procurement of the macro definition. Thereafter, this case is processed as if the macro name had been found in the macro name dictionary. In each case, the current LOF entry is supplemented with the appropriate directive code and machine instruction code information.

Macro name dictionary items are kept in working segment 2, but are located through a different hash table from the one used for ordinary symbols. This permits macro names to duplicate location symbols without confusion and with a minimum of interference for the user of the main dictionary

Error Check: Undefined mnemonic code.

SUBOP -- Substitute into Operation Code Field (CEVSP)

This routine isolates the operation field of a source statement and performs the substitution of any variable symbols that may occur within the field. The operation code resulting from the substitution determines the further processing of the source statement. (See Chart AF.)

Entry Point: CEVSP

Calling Sequence: INVOKE ASUBOP
... error return
... normal return

Routines Called: SSCAN2

Exit: Normal

OPERATION: The current statement is scanned for the first blank. The first nonblank following the first blank is taken as the start of the operation field. The field is scanned for ampersands or blanks. If an ampersand is found, the SSCAN2 entry to SSCAN is initialized and called to perform the character substitution. The substituted operation mnemonic is built up in working storage. Upon return from SSCAN2 the scan for ampersands or blanks is resumed since variable symbols may be concatenated to form a single mnemonic. The occurrence of a blank terminates the scan. Any characters from the original field not moved during the variable symbol substitution are concatenated with the contents of the work area, and the length of the result is checked for validity. Valid results are left-justified on a field of blanks at location OP, where the resultant mnemonic is available for later use.

CATOP -- String Substitution Control (CEVCP)

This routine controls the type and amount of parameter and variable symbol substitution applied to the current source statement. It is called before the statement is delivered to the components of STAN for processing. (See Chart AG.)

Entry Point: CEVCP

Routines Called: BASCAN, SSCAN

Exits: Normal
Error detected in SSCAN

OPERATION: Substitution is performed arbitrarily on the operation code field by GETOP. This routine examines the operation as determined by GETOP, applying the following rules:

- If GBLx or LCLx, no further substitution is attempted.
- If AIF or SETB:

String substitution of parameters and SETA and SETC symbols is performed unconditionally in the operand field.

SETB symbols are substituted in the operand if the variable symbol is encountered within apostrophes.

When not within apostrophes, SETB symbols are substituted in the operand when the adjacent characters indicate concatenation.

- If SETx, substitution is not performed in the name field.

In all other cases, substitution is performed in both name and operand fields unconditionally. Whenever substitution actually results in character string replacement on a statement, a new version of the statement reflecting the substitution is produced to replace the original line for all subsequent processing.

After substitution, this routine calls BASCAN to analyze the contents of the name field and leaves the results for later analysis by components of STAN. This routine also determines the start of the operand field, posts the increment in the current LOF entry, and sets BSSCAN to the location of the start of the operand field.

MIP -- Machine Instruction Operand Scan (CEVMP)

This routine scans the operands of machine instructions and checks for valid operand fields and correct formatting. It assumes that BSSCAN is set by the caller to the address of the first character of the operand. (See Chart AH.)

Entry Points: CEVMP1 CEVMP7
 CEVMP2 CEVMP8
 CEVMP3 CEVMP9
 CEVMP4 CEVMPA
 CEVMP5 CEVMPB
 CEVMP6

Calling Sequences: INVOKE AOPxxx
 xxx may have the following values:
 RR1 RX1 RS2 SS1
 RR2 RX2 SI1 SS2
 RR3 RS1 SI2

Routines Called: DIAG, EVAL

Exit: Normal

OPERATION: For each type of instruction there is a main stem of the subroutine which checks to see that the required operand fields are present and that these are properly delimited. These main stems then call subroutines internal to this processor to examine the components of the operand field; these, too, are checked to see that required parts are present and that they are correctly delimited.

Error Checks:

- Major operand fields must be separated by a comma.
- All major operand fields must be present.

- The end of an instruction operand must be delimited by a blank.

- Expressions with a left parenthesis must be delimited on the right by a right parenthesis.
- Expressions within parenthesis not separated by operators must be separated by a comma.
- Expressions representing registers must not exceed an absolute value of 15.
- Expressions representing lengths must not exceed absolute values of 16 or 256, depending upon whether they are represented by four or eight bits, respectively, in the individual instruction.
- To be valid an expression representing a base register must be absolute, null, or indeterminant.
- To be valid an expression representing a length must be absolute, null, or indeterminate.
- Nothing can be stored into a literal.
- An expression representing a shift value must be absolute or indeterminate.

BASCAN -- Basic Source Language Scan (CEVBS)

This routine scans part of a source language statement in order to identify the syntactic components of the language that appear within that part of the statement. It assumes that certain storage areas have been set by the caller. These storage areas are:

BSSCAN Address where scanning is to begin.

BSMAX Address of last byte in source statement.

BSMI Indicates scan mode.

Hex Value	Mode	Action
00	Blank	Resume scan at next nonblank character.
12	Period	Previous delimiter was a period; interpret the new term as a fraction if it is numeric or as a sequence symbol if it is non-numeric.

		Type	Hex value
14	Character-string	Previous delimiter was a single quote; resume scan at current character, in character - string mode.	Blank 00 Term 01 Delimiter only 02 Character-string 04 Attribute I 0C Attribute S 0D Attribute T 0E Attribute L 0F
40	Name field	Causes scan to recognize commentary and/or blanks in the first scanned column(s).	Symbol 10 Sequence symbol 11 Variable symbol 12 Location counter 13
80	Continuation	Previous delimiter was a nonnumeric character after a decimal integer string; use previous delimiter as the first character of the new term.	Self-defining term Decimal 28 Hexadecimal 29 Binary 2A Character 2B Attribute K 4C Attribute N 4D

Any other Term
delimiter value
(see BSMI
output
values
below)

Previous delimiter was a comma, parenthesis, equal sign or arithmetic operator; resume scan at current character, initialized for a new term.

FAL Indicates whether expression is arithmetic (0) or logical (1).

(See Chart AI.)

Entry Point: CEVBS

Calling Sequence: INVOKE ABSCAN

Routines Called: DIAG

Exit: Normal

OPERATION: This routine uses and updates a pointer and a mode flag as it operates. The pointer is the basic scan index (BSSCAN) and is the virtual address of the byte in the source statement at which scanning is to start or resume. The mode flag is the basic scan mode indicator (BSMI) and is generally the right delimiter of the previously scanned field. Routines that call this routine may set both BSSCAN and BSMI to achieve specially desired results from the scanning process. Otherwise, this routine uses the values that it placed in BSSCAN and BSMI to continue scanning in an orderly fashion.

Return is made to the caller with the following fields set:

BSBEG Location of scanned field
BSLNTH Length of scanned field
BSTYPE Type of scanned field

BSMI Delimiter of scanned field

Delimiter	Hex Value
Blank	00
'	02
)	04
(06
=	10
.	12
,	14
&	18
End of statement	20
Name field	40
Nonnumeric character	80
+	F4
-	F6
*	F8
/	FA

BSERR Error encountered by BASCAN

Error	Hex Value
No errors	00
Invalid character in statement	01
Name or operand field contains invalid character	02
Field improperly delimited	04
Invalid binary self-defining term	08
End of statement encountered before processing completed	10
Invalid use of asterisk (*)	20

AGO/AIF -- AGO/AIF Instruction Scan (CEVGO)

This routine examines the name and operand fields of AGO and AIF instructions for syntactical correctness. It assumes the caller has previously called BASCAN to examine the name field of the instruction being processed and that all storage areas affected by this call are unchanged except for BASCAN which the caller has set to the address of the first character in the operand field. (See Chart AJ.)

Entry Points: CEVGO1, CEVGO2

Calling Sequence: INVOKE AAGO
 INVOKE AAIF

Routines Called: BASCAN DLKT
 DIAG EVAL
 DLKM

Exit: Normal

OPERATION: The AGO entry to this routine corresponds to a true condition in the AIF processing.

If the name field of either instruction contains neither blanks nor a sequence symbol, a warning is given, and the name field is ignored. The AIF processing begins by calling EVAL in the logical mode, to determine whether the operand represents a true or false value. If false, the remainder of the statement is checked for syntactical correctness, and control returns to STAN.

If a true value, the sequence symbol is examined. Depending upon the macro level, the symbol is looked up in either the main dictionary or the temporary dictionary. If it is found in a dictionary, the obtain next source statement routine (REED) input source switch is pushed down, and a copy mode is established for the source statement defined by the sequence symbol.

If the symbol has not yet been defined, a bypass mode is established for STAN. This mode causes all processing to be suppressed until a statement bearing the desired sequence symbol appears or until an END statement or end of file occurs. MEND will also terminate this processing if a macro is currently being expanded.

Error Checks:

- Invalid symbol in name field.
- Sequence symbol missing in operand.
- Invalid statement format.
- Improper delimiter.

ANOP -- ANOP Instruction Scan (CEVAN)

This routine scans the ANOP instruction for a valid name field. It assumes the caller has previously called BASCAN to examine the name field of the instruction being processed and that storage areas affected by that call are unchanged. (See Chart AK.)

Entry Point: CEVAN

Calling Sequence: INVOKE AANOP

Routines Called: DIAG

Exit: Normal

OPERATION: The name field is checked for the presence of a sequence symbol; a diagnostic is generated if one is not present. Control is then returned to the caller.

CCW -- CCW Instruction Scan (CEVCW)

This routine examines CCW instructions for valid operand fields and correct format. It assumes that the caller has called BASCAN to examine the name field of the instruction being processed, and that all storage areas affected by the call are unchanged except BSSCAN, which the caller has set to the first character in the operand. (See Chart AK.)

Entry Point: CEVCW

Calling Sequence: INVOKE ACCW

Routines Called: DEFSYM EVAL
 DIAG SLIT

Exit: Normal

OPERATION: Upon entry, DEFSYM is called. A test is then made to determine the mode in which the assembly is being processed. Batch mode causes the operand to be scanned for the presence of literals and an exit to the caller. If in conversational mode, each operand field is then examined; all must be present.

The first operand must be absolute, null, or indeterminate, and must be delimited by a comma. If the latter case is not true, it is assumed that the remainder of the operands are missing.

The second operand may be any type of expression and must be delimited by a comma. If the latter case is not true, it is assumed that the remainder of the operand fields are missing.

The third operand field is treated exactly the same as the first operand.

The fourth operand field is treated like the first, except that the delimiter must be a blank.

Error Checks:

- All operands must be present.

- All operands except the last must be delimited by a comma.
- The last operand must be delimited by a blank.
- The first, third, and fourth operands must be absolute, null, or indeterminate.

CNOP -- CNOP Instruction Scan (CEVCN)

This routine examines the CNOP instruction operand field for valid expressions and correct format. It assumes that the caller has previously called BASCAN to examine the name field of the instruction being processed and that all storage areas affected by this call are unchanged except for BSSCAN, which the caller has set to the address of the first character in the operand field. (See Chart AL.)

Entry Point: CEVCN

Calling Sequence: INVOKE, ACNOP

Routines Called: DIAG, EVAL

Exit: Normal

OPERATION: Operation is different for conversational and batch mode.

Conversational Mode: The name field is checked to make sure it is blank or contains a sequence symbol. If the name is invalid, it will be ignored. The first operand field is then examined by the expression evaluator. Only absolute, null, or indeterminate expressions are valid, and the absolute value must be 0, 2, 4, or 6. This operand field must be delimited by a comma, which is followed by the second operand field.

The second operand field must also be an absolute or indeterminate value to be valid. Allowable values for an absolute expression are 4 or 8. This second expression must be delimited by a blank.

Allowable combinations of these absolute values are:

0,4	2,8
2,4	4,8
0,8	6,8

Batch Mode: The name field is examined, SLIT is called, and control is returned to the caller.

Error Checks:

- Two operand fields must be separated by a comma.

- The second operand must be delimited by a blank.

- The first operand field must be absolute, null, or an indeterminate expression; if absolute it must have a value of 0, 2, 4, or 6.

- The second operand field must be indeterminate or absolute; if absolute, it must have a value of 4 or 8.

- Valid combinations of absolute first and second operand fields are:

0,4	2,8
2,4	4,8
0,8	6,8

- The name field must be blank or contain a sequence symbol.

CXD -- CXD Instruction Scan (CEVCX)

This routine examines the CXD instruction and validates the name field, if any is specified. It assumes that the caller has previously called BASCAN to examine the name field of the instruction being processed, and that storage areas used by BASCAN are unchanged. (See Chart AL.)

Entry Point: CEVCX

Calling Sequence: INVOKE ACEVCX

Routines Called: DEFSYM

Exit: Normal

OPERATION: This routine is entered from STAN after calling BASCAN. DEFSYM is called to build a dictionary item if necessary. Upon return from DEFSYM, the current LOF entry is completed, and a constant item is built. Return is then made to the caller.

SECT -- Control Section Instruction Scan (CEVCT)

This routine checks the control section instructions for validity. It assumes that the caller has previously called BASCAN to examine the name field of the instruction being processed and that all storage areas affected by this call are unchanged except for BSSCAN, which the caller has set to the address of the first character in the operand field. (See Chart AM.)

Entry Points: CEVCT1 CEVCT4
CEVCT2 CEVCT5
CEVCT3

Calling Sequences: INVOKE Axxxxx
xxxxx may be: CSCOM CSDCT
CSECT CSPCT
CSTRT

Routines Called: BASCAN DLPM
DIAG EVAL

Exit: Normal

OPERATION: There is a separate entry point for each type of control section instruction. At this point the code representing the specific type of instruction is set, and pointers to the first control section dictionary item and first PSECT dictionary item are set, when appropriate. The name field has been collected by BASCAN. Unnamed PSECTS and DSECTS are considered errors; unnamed CSECTS and COMS are valid. The name field symbol is looked up in the main dictionary; a look-up for a name of blanks or binary zeros will occur for unnamed COMS and CSECTS, respectively. If the symbol has not been previously defined, dictionary and GSM entries are created, and the second part of the routine begins. If an entry for the symbol was found, the type of dictionary item and the type of current statement are compared. If the two are equivalent, a GSM entry is created, and the routine continues.

If the current instruction is macro generated, the control sections are reordered, if necessary. If the symbols do not designate the same control section, there is either a duplicate symbol or incompatible definition of control sections. In the first case, the routine is merely halted and control returned to the caller. In the latter case, an "incompatible control sections" diagnostic is issued at this point. The label is set to blanks, and, if the instruction is not a PSECT or DSECT (unnamed PSECTS or DSECTS are errors), the dictionary will be searched again. Normal processing follows. If this label is determined to be conflicting, an exit from the routine is made. Each valid control section name is compared with the program module name. Duplication results in a warning message to the programmer. Also, each control section name is checked to make sure certain symbol conventions are not violated.

If the instruction is a START, an ORG statement is generated. The START operand is evaluated as if it were that of an ORG instruction, after which control is returned to the caller. For other instructions, BASCAN is called to inspect the operand for valid fields. The operand fields, if specified, are attributes which are PUBLIC, READONLY, VARIABLE, PRVLGD, or SYSTEM. The dictionary item is updated with the control section type and any valid

attributes. The operand fields are to be delimited by a comma or blank.

The label for a blank, unnamed CSECT is carried as two full words of binary zero -- X'0000000000000000'. The label for a blank, unnamed COM is carried as two full words of blanks -- X'4040404040404040'.

Error Checks:

- The operand field is not delimited by comma or blank.
- Unnamed DSECT.
- Unnamed PSECT.
- Duplicate symbols.
- Incompatible control section statements.
- Attribute other than PUBLIC, READONLY, VARIABLE, PRVLGD, or SYSTEM.
- START not the first control section statement.
- More than five attributes.
- The name field must contain blanks or a valid symbol.
- Control section name may not duplicate module name.
- Control section name must not violate symbol conventions.

COPY -- COPY Instruction Processor (CEVCY)

This routine checks the validity of the COPY operand, and if required, insures the element is retrieved from the COPY library. (See Chart AN.)

Entry Point: CEVCY

Calling Sequence: INVOKE ACOPY

Routines Called:

- Internal - DIAG
- External -
 - CEVMLA Find element in library
 - CEVMLB Retrieve lines from library

Exit: Normal

OPERATION: This module is called by the statement analyzer or macro reference processor when the statement to be processed is a COPY instruction. The contents of the operand field are collected; the desired

element is then retrieved from the library and copied into working storage. A diagnostic is given if the element is absent from the library. The source statements are chained together with the standard source statement control bytes. The input switch of the REED routine is then pushed down and set to retrieve forthcoming statements from the copied stack.

Error Checks:

- Invalid or missing operand.
- Missing library element.
- Nested COPY statements.

DC/DS -- DC/DS Instruction Scan (CEVDC)

This routine is called by the statement analyzer (STAN) when the statement to be processed is a DC, DS, or DXD instruction. The routine calls upon the constant scan (CSCAN) to analyze the constant and to prepare a value for it. The resulting constant item is then associated with the LOF entry for the statement, and the definition of any name defined by the constant is completed. (See Chart AO.)

This routine assumes that the caller has previously called BASCAN to examine the name field of the instruction and that all storage areas affected by that call are unchanged except BSSCAN, which the caller has set to the address of the first character in the operand field.

Entry Points: CEVDC1, CEVDC2, CEVXD

Calling Sequence: INVOKE ADC
INVOKE ADS

Routines Called: CSCAN, DEFSYM, DIAG

Exit: Normal

OPERATION: This routine calls DEFSYM to define any symbol that may appear in the name field of the statement. If the instruction is a DXD, it is marked as such before calling DEFSYM. A diagnostic is issued if the DXD instruction being processed is missing a name. CSCAN is called to analyze the operand and to prepare a constant value item. The routine associates the constant item with the current LOF entry and completes the dictionary entry for the name defined by the constant, if any. A DXD instruction will cause an entry to be made on the external dictionary chain. If the constant contains multiple operands, a series of calls is made to CSCAN, and a specially flagged LOF entry is constructed for each of the multiple operands.

EJECT -- EJECT Instruction Scan (CEVEJ)

This routine checks the EJECT instruction for correct format. It assumes that the caller has previously called BASCAN to examine the name field of the instruction and that the storage areas associated with that call are unchanged. (See Chart AP.)

Entry Point: CEVEJ

Calling Sequence: INVOKE AEJECT

Routines Called: DIAG

Exit: Normal

OPERATION: The name field is checked to see that it contains either blanks or a sequence symbol. If so, control is returned to the caller. If not, a diagnostic is given and control is returned to the caller.

END -- END Instruction Scan (CEVND)

This routine checks the END instruction for correct formatting and valid operation fields. (See Chart AP.)

Entry Point: CEVND

Calling Sequence: INVOKE AEND

Routines Called: DIAG, EVAL

Exit: Normal

OPERATION: If the assembly is in batch mode, the name field is checked, the END indicator set, and control returned to the caller. In conversational mode, the name field is checked for blanks or a sequence symbol, and, if in error, is ignored. The operand field is then examined by EVAL to make sure the value of the expression is other than absolute, literal, or error. The macro level indicator is checked to make sure the statement did not occur within a macro. After the end indicator is set, control is returned to the caller.

Error Checks:

- An error occurs if the operand field has an absolute or error value, or is a literal.
- If the macro level indicator is greater than zero, the statement has occurred within a macro which is an error.
- The name field should be blank, or contain a sequence symbol.
- The single operand should be delimited by a blank.

ENTRY -- ENTRY Instruction Scan (CEVEY)

This routine scans the ENTRY instruction for correct format and valid operation fields. It assumes that the caller has called BASCAN to examine the name field of the instruction, and that all storage areas affected by that call are unchanged except BSSCAN, which the caller has set to the address of the first character in the operand field. (See Chart AQ.)

Entry Point: CEVEY

Calling Sequence: INVOKE AENTRY

Routines Called: BASCAN, DIAG

Exit: Normal

OPERATION: A GSM entry is constructed for the statement, and the name field is checked for blanks or a sequence symbol. If the name field is found to be invalid, it is ignored. BASCAN is called to collect and examine each operand field. It is expected that there is at least one operand field, and each field is delimited by either a comma or a blank. Each entry is then checked to determine whether or not it violates any symbol conventions. These conventions are a function of the attributes of the control section in which the entry occurs.

Error Checks:

- At least one operand is expected to be present.
- Entry point must conform to certain symbol conventions.
- Fields must be delimited by either a comma or blank.
- Name field should be blank or contain a sequence symbol.

EQU -- EQU Instruction Scan (CEVQU)

This routine examines the EQU instruction for syntactical correctness. It assumes that the caller has previously called BASCAN to examine the name field of the instruction and that all storage areas affected by that call are unchanged except BSSCAN, which the caller has set to the address of the first character in the operand field. (See Chart AQ.)

Entry Point: CEVQU

Calling Sequence: INVOKE AEQU

Routines Called: DIAG DLPM
DLKM EVAL

Exit: Normal

OPERATION: If the name field is missing, the LOF entry is made commentary and exit is made to the caller. Otherwise, EVAL is called to process the three possible operand fields for validity. Absolute or complex dictionary items are made for the name field symbol when the operand is represented by an absolute or complex expression. The length and type operands are checked for validity. If an error is detected, a diagnostic is issued and the length and type values are determined as if they were defaulted. If the expression in the operand is simply relocatable or indeterminate, a transitive item is constructed for the symbol in the name field, and the LOF entry is flagged for attention during Phase IIB.

Error Checks:

- Expression value invalid for field.
- Missing operand field.
- Missing name field.
- Invalid expression type.
- Duplicate symbol in name field.
- Invalid name field.
- Field improperly delimited.
- Invalid operand field.

EXTRN -- EXTRN Instruction Operand Scan (CEVXN)

This routine examines the EXTRN instruction operand for valid operand fields and correct formatting. It assumes that the caller has previously called BASCAN to examine the name field of the instruction and that all storage areas affected by that call are unchanged except BSSCAN, which the caller has set to the address of the first character in the operand field. (See Chart AR.)

Entry Point: CEVXN

Calling Sequence: INVOKE AEXTRN

Routines Called: BASCAN, DIAG, DLPM

Exit: Normal

OPERATION: The name field is examined for blanks or a sequence symbol. If the name field is invalid, a diagnostic is issued, the name is ignored, and processing continues. BASCAN is called to examine the operand. If there is no operand, a warning

diagnostic is generated. Finding a symbol results in calling DLPM. If the symbol has been previously defined, an error has occurred; the programmer is warned that the symbol will not be treated as external. Otherwise, a dictionary item is created. In either event, control is then returned to the caller.

Error Checks:

- The operand is blank.
- There is no symbol following a comma.
- The symbol has been previously defined.
- A symbol is delimited by something other than a comma or blank.
- The name field should be blank or contain a sequence symbol.

GBLx/LCLx -- Global/Local Symbol Instruction Scan (CEVGL)

This routine scans Global and Local instruction operands and checks the operand fields for valid expressions and correct format. It assumes that the caller has previously called BASCAN to examine the name field of the instruction and that all storage areas affected by that call are unchanged except BSSCAN, which the caller has set to the address of the first character in the operand field. (See Chart AS.)

Entry Points: CEVGL1 CEVGL5
CEVGL2 CEVGL6
CEVGL3 CEVGL7
CEVGL4

Calling Sequences: INVOKE Axxxx

xxxx may have the following values:

GBLA LCLA
GBLB LCLB
GBLC LCLC

Routines Called: BASCAN DLPM
DIAG DPUT
DLKT EVAL

Exit: Normal

OPERATION: This will vary depending on which phase is calling the routine.

Phase I: Each type of symbol instruction has its own entry point. Here the dictionary item type, code, and length are stored.

BASCAN is called to collect the first operand field; only variable symbols are

allowed as operand fields. If the macro level indicator is greater than zero, the temporary dictionary is searched for the variable symbol. A find is considered an error because a symbol can be defined only once. If the symbol is not found, and it is a local symbol, it is inserted into the temporary dictionary. If the symbol was not found in the temporary dictionary, and it is a global symbol, a search of the main dictionary is initiated. No find causes an item to be inserted in both the main and temporary dictionaries; a valid find results in an insertion in the temporary dictionary. The processing advances to the point where the delimiter is checked.

If the macro level was zero, the main dictionary is searched for the symbol. A find indicates an error; no find results in an item being inserted in the main dictionary. The delimiter of the operand field is checked.

If the delimiter of the variable symbol is a left parenthesis, there is a subscript to be evaluated. The subscript must be an absolute value less than or equal to 255; the valid value is inserted into the dictionary item. Anything greater than 255 is set to 255, and a diagnostic is generated. The delimiter of the subscript is checked to be a right parenthesis. The next character must be a comma or a blank. A comma causes the processing to return to the point where basic scan is called. A blank causes exit to the caller. This cycle is continued until a blank delimiter is encountered or error condition prevents further processing.

If the operation is global, and it is not Phase IIA, a GSM entry is created.

Phase IIA: During Phase IIA, this module performs two functions. The first is to process those global and local symbols occurring in macro expansions. These are processed as in Phase I, except no GSM entry is made for global symbols. The second Phase IIA function this module performs is to reprocess those global instructions seen in Phase I. The result of this reprocessing is to reset the symbols to their initial values so the correct values will be associated with the symbols during macro expansions.

Error Checks:

- Operand field must contain variable symbol.
- Only one definition per symbol allowed in one dictionary.
- Delimiter of variable symbol must be left parenthesis, comma, or blank.

- Subscript expression must be absolute.
- Absolute value of subscript must be less than or equal to 255.
- Subscript must be delimited by right parenthesis.
- Name field should be blank.

ICTL -- ICTL Instruction Scan (CEVIC)

This routine scans ICTL instruction operand for valid operand fields with correct formatting. It assumes that the caller has previously called BASCAN to scan the name field of this instruction and that all storage areas affected by that call are unchanged except for BSSCAN, which the caller has set to the address of the first character in the operand. (See Chart AT.)

Entry Point: CEVIC

Calling Sequence: INVOKE AICTL

Routines Called: DIAG, EVAL

Exit: Normal

OPERATION: To be valid, an ICTL must be the first source program statement. An invalid occurrence of an ICTL causes the statement to be made commentary.

To be valid, the name field should be blank or contain a sequence symbol. If this is not the case, the name is ignored.

The first operand field, indicating the begin column, must be present. It is checked to see that it is greater than zero and less than or equal to 40. If the absolute value is valid, it is stored in BCOL. An error in the begin column causes the statement to be ignored.

The second operand field is evaluated to determine whether it is present. Its absence causes the end column to be set to its assumed value of 71. If the operand is present, it must represent an absolute value greater than 40 and less than or equal to 80; otherwise, it is considered an error, and the end column is assumed to be 71. If the value is 80, it is assumed that there will be no continuation cards.

The third operand field is examined. If the field contains an operand, and the end column is 80, an error occurs and continuation cards are not allowed. In event of any other type of error in the field, the continue column is assumed to be 16. Should this conflict with the begin column, continuation cards will be ignored. A blank field or a value of 80 for the end

column indicates that continuation cards are not allowed.

If the field contains an absolute expression, the value must be greater than that designated as the begin column, and must be greater than 1 and less than or equal to 40.

Each operand field is checked to be delimited by either a comma or blank. Only one ICTL statement is honored during one assembly, and it must precede all other source statements.

Error Checks:

- There can be only one ICTL statement per assembly.
- The first operand, which designates begin column, must be present.
- All operands, if present, must be in form of absolute values.
- The begin column must be greater than 0 and less than or equal to 40.
- The end column must be greater than 40 and less than or equal to 80.
- If the end column is 80, there can be no continue column designation.
- The continue column must be greater than 1, greater than the begin column, and less than or equal to 40.
- Each field must be delimited by a comma or blank.
- The name field should be blank or contain a sequence symbol.

ISEQ -- ISEQ Instruction Scan (CEVIQ)

This routine examines the ISEQ instruction operand fields for valid expressions and correct format. It assumes that the caller has previously called BASCAN to scan the name field of this instruction and that all storage areas affected by that call are unchanged except for BSSCAN, which the caller has set to the address of the first character in the operand. (See Chart AU.)

Entry Point: CEVIQ

Calling Sequence: INVOKE AISEQ

Routines Called: DIAG, EVAL

Exit: Normal

Error - Model statement is ISEQ or ICTL, or assembler instruc-

tion other than END used as macro.

OPERATION: If BASCAN found an error in the ISEQ name field, the name field is ignored. If BASCAN did not find an error, the name field is checked for a blank or a sequence symbol; anything else results in a diagnostic message, and processing continues.

The first operand field is examined by EVAL. A blank operand indicates that sequence checking is to terminate; to indicate this, the areas indicating the left and right column boundaries for the sequence check are set to zero. An absolute expression designates the left column (or begin column) of sequencing. This value is compared with the end column. To be valid, the value must be greater than the end column value +1, or less than the begin column and greater than zero. The delimiter is checked to make sure it is a comma.

The second operand representing the right boundary of the columns to be sequence checked is evaluated. It must be an absolute value equal to or greater than the left column. If the left column of the sequence area is less than the begin column value, the right column of the sequence area must also be less than the begin column value. If the left column of the sequence area is greater than the end column, the right column value may not exceed 80. Its delimiter is checked to make sure it is a blank.

Any error in the statement results in a cancellation of the sequence check; LCOL and RCOL are set to zero, and control returned to the caller.

Error Checks:

- Operand fields should be blank or absolute.
- Operand fields should be separated by a comma.
- The value of the left column should be at least 2 greater than ECOL, or less than BCOL and greater than zero.
- The second operand is delimited by a blank.
- The right column should be equal to or greater than the left column and less than or equal to 80, or 1 or greater and less than BCOL.
- The name field should be blank or contain a sequence symbol.

LTORG -- LTOrg Instruction Scan (CEVLG)

This routine checks the LTOrg instruction for correct format. It assumes that the caller has previously called BASCAN to scan the name field of this instruction and that the storage areas affected by that call except BSSCAN are unchanged. (See Chart AU.)

Entry Point: CEVLG

Calling Sequence: INVOKE ALTORG

Routines Called: DEFSYM

Exit: Normal

OPERATION: DEFSYM is called to construct a relocatable value item from the symbol, if one is designated in the name field. A GSM entry is created, and control is returned to the caller.

MACRO -- MACRO Instruction Scan (CEVMC)

This routine checks the MACRO instruction for correct format. It assumes that the caller has previously called BASCAN to scan the name field, and that all storage areas affected by that call except BSSCAN are unchanged. (See Chart AV.)

Entry Point: CEVMC

Calling Sequence: INVOKE AMACRO

Routines Called: DIAG

Exit: Normal

OPERATION: The name field is checked to make sure it is blank; if it is not, a diagnostic is printed. The macro definition switch is then set to 1, and control is returned to the caller.

MEND/MEXIT -- MEND/MEXIT Instruction Scan (CEVMX)

This routine examines MEND and MEXIT instructions for correct format. It is phase oriented, in that the MEXIT entry can only occur during Phase IIA. The routine either cancels macro definition mode or causes the macro expansion mechanism to return to the previous macro level. (See Chart AV.)

This module assumes that for a MEXIT instruction the caller has previously called BASCAN to examine the name field, and that all storage parameters affected by that call except BSSCAN are unchanged.

Entry Points: CEVMX1, CEVMX2

Calling Sequence: INVOKE AMEND
INVOKE AMEXIT

Routines Called: BASCAN, DIAG

Exits: Normal - CEVST1
Error - Original exit

OPERATION: Each instruction scan has a separate operation.

MEND: In Phase I the name field is analyzed; invalid entries are diagnosed and ignored. This check is not made during Phase IIA. If the macro definition mode is set (MDS switch = 2), it is canceled (MDS switch is set to zero), and an immediate return is made. This condition prevails during the processing of macro definitions in either Phase I or IIA. If the macro definition mode is not set, MEND executes identically with MEXIT.

MEXIT: The name field of the instruction is examined; invalid entries are diagnosed and ignored. If the macro level is zero, MEXIT has occurred out of context in the source program. A diagnostic is given, and the statement is ignored. If the macro level is greater than zero, the space occupied by the current macro level dictionary is reclaimed (by resetting AWORK1 address pointer). The macro level (MLVL) is reduced by one, and the location of the LOF entry for the statement at which processing stopped on the preceding macro level is reinstated in the REED input switch. If the macro level has been reduced to zero, the REED input switch is popped up to its previous mode.

Error Checks:

- Name field improperly delimited.
- Name field contains invalid symbol.
- MEXIT instruction invalid outside macro.

MNOTE -- MNOTE Instruction Scan (CEVMN)

This routine scans the operand of MNOTE instructions, checking for valid operands and correct format. It assumes that the caller has previously called BASCAN to examine the name field of this instruction, and that all storage areas affected by that call are unchanged except BSSCAN, which has been set to the address of the first character in the operand. (See Chart AW.)

Entry Point: CEVMN

Calling Sequence: INVOKE AMNOTE

Routines Called: BASCAN, DIAG, EVAL

Exit: Normal

OPERATION: The first operand is examined by EVAL. Null, absolute, or indeterminate expressions, as well as an *, are considered valid. The delimiter is checked to make sure that it is a comma, and that it is followed by a character string. (If the operand contains only a character string, the MNOTE is treated as diagnostic with severity code of zero.) If the first operand was an *, the character string is treated as a comment and is limited to a length of 226 characters. Otherwise, the character string is considered a diagnostic message and causes a special call to the diagnostic processor. In this case, a maximum of 100 characters is allowed. Control is returned to the caller.

Error Checks:

- First operand, if present, other than absolute, null, or asterisk.
- First operand, if present, not delimited by a comma.
- Something other than a blank or sequence symbol in name field.
- Diagnostic character string greater than 100.
- Comment character string greater than 226.
- If only one operand is present, it represents something other than a character string.
- If the first operand is present, the second operand represents something other than a character string.

ORG -- ORG Instruction Scan (CEVRG)

This module evaluates the operand of ORG instructions and determines whether it is valid and the format correct. It assumes that the caller has previously called BASCAN to scan the name field of this instruction, and that all storage areas affected by that call are unchanged except BSSCAN, which has been set to the address of the first character in the operand. (See Chart BA.)

Entry Point: CEVRG

Calling Sequence: INVOKE AORG

Routines Called: DIAG, EVAL

Exit: Normal

OPERATION: The name field is checked for a blank or a sequence symbol. If the name field is not valid, a diagnostic message is issued, the name is ignored, and processing continues. A test is then made to determine the mode in which the assembly is being processed; batch mode results in immediate exit to the caller.

In conversational mode, EVAL is called to examine the operand. Absolute, null, or relocatable values are valid. The delimiter is checked to verify that it is a blank, and control is returned to the caller.

Error Checks:

- Expression must be absolute and nonnegative, null, or indeterminate.
- The field must be delimited by a blank.
- The name field should be blank or contain a sequence symbol.

PRINT -- PRINT Instruction Operand Scan (CEVPR)

This module examines the operand of PRINT instructions for correct formatting and valid contents. It assumes that the caller has previously called BASCAN to scan the name field of this instruction and that all storage areas affected by that call are unchanged except BSSCAN, which has been set to the address of the first character in the operand. (See Chart BA.)

Entry Point: CEVPR

Calling Sequence: INVOKE APRINT

Routines Called: BASCAN, DIAG

Exit: Normal

OPERATION: If BASCAN found an error in the PRINT name field, the name field is ignored. If BASCAN did not find an error, the name field is checked for a blank or a sequence symbol; anything else results in a diagnostic message, and processing continues.

A GSM entry is created for the PRINT statement. The operand field is examined to see that no more than three of the following options appear, and that none are contradictory or repetitive: ON, OFF, GEN, NOGEN, FULLGEN, DATA, NODATA. An option must be delimited by a blank or comma.

Codes for valid options are entered in the current logical order file entry.

Error Checks:

- A blank operand.
- A comma followed by a blank.
- An option not delimited by a comma or blank.
- Something other than ON, OFF, GEN, NOGEN, FULLGEN, DATA, or NODATA specified as an operand field.
- More than three options.
- Options which contradict or repeat one another.
- The name field should be blank or contain a sequence symbol.

PUNCH -- PUNCH Instruction Scan (CEVPH)

This routine issues a warning when a PUNCH instruction is encountered.

Entry Point: CEVPH

Calling Sequence: INVOKE APUNCH

Routines Called: DIAG

Exit: Normal

OPERATION: The PUNCH instruction is allowed to maintain compatibility with OS/360. The LOF entry for the PUNCH instruction is changed so it will result only in a printed line in the listing. A diagnostic is printed to warn the user:

WARNING: INSTRUCTION PRODUCES LISTING ONLY.

REPRO -- REPRO Instruction Scan (CEVRE)

This routine issues a warning when a REPRO instruction is encountered.

Entry Point: CEVRE

Calling Sequence: INVOKE AREPRO

Routines Called: DIAG

Exit: Normal

OPERATION: The REPRO instruction is allowed to maintain compatibility with OS/360. The LOF entry for the REPRO instruction is changed by another module so it will result only in a printed line in the

listing. A diagnostic is printed to warn the user:

WARNING: INSTRUCTION PRODUCES LISTING ONLY.

SETX -- SET Instruction Scan (CEVSE)

This routine processes SETA, SETB, and SETC instructions. It assumes that the caller has previously called BASCAN to scan the name field of this instruction and that all storage areas affected by that call are unchanged except BSSCAN, which has been set to the address of the first character in the operand. (See Chart BB.)

Entry Points: CEVSE1, CEVSE2, CEVSE3

Calling Sequence: INVOKE SETx
x may be: A, B, C

Routines Called: BASCAN, DIAG, EVAL

Exit: Normal

OPERATION: This routine is called when the statement to be processed is a SETA, SETB, or SETC instruction. The symbol in the name field is checked for validity, and set to the value of the expression which appears in the operand. Repetitive definition of the same symbol is permitted. The permanent dictionary is used when the macro level is zero; the current macro level dictionary is used when the macro level exceeds zero.

SETA Instruction: If the name field is unsubscripted, EVAL is called to process the operand. Only absolute arithmetic values are acceptable. The value is placed in the appropriate dictionary item. If the name field is subscripted, the subscript is checked for validity, and a subscript trailer is added to the dictionary item, if required. The operand is then evaluated, checked for validity, and inserted into the subscript trailer. The value is also inserted in the logical order file entry.

SETB Instruction: If the name field is unsubscripted, EVAL is called in the logical expression mode to process the operand. The results must be logically true or false to be valid. The value bit in the dictionary item is set or reset as required. If the name field is subscripted, the subscript is checked for validity. EVAL is called to evaluate the logical expression, and the bit in the dictionary item which represents the subscripted value is set or reset.

SETC Instruction: If the name field is unsubscripted, BASCAN is called to process the operand. Only character strings or the

type attributes are acceptable. The character string must be eight characters or fewer. If valid, the value is placed in the dictionary item. When the name field is subscripted, the value is checked for validity, and a subscript trailer is added to the dictionary item, if required. The operand is then scanned, checked for validity, and inserted into the subscript trailer. The value and length of the symbol are also inserted in the logical order file entry.

Error Checks:

- An invalid name field.
- A missing subscript.
- An invalid subscript.
- A character string too long.
- An invalid expression type.
- An invalid value for the field.
- An invalid operand.
- Field improperly delimited.
- Statement incompatible with previous definition.

SPACE -- SPACE Instruction Scan (CEVCE)

This routine examines SPACE instructions for syntactical correctness. It assumes that the caller has previously called BASCAN to scan the name field of this instruction and that all storage areas affected by that call are unchanged except BSSCAN, which has been set to the address of the first character in the operand field. (See Chart BC.)

Entry Point: CEVCE

Calling Sequence: INVOKE ASPACE

Routine Called: DIAG, EVAL

Exit: Normal

OPERATION: The name field is checked for the presence of blanks or a sequence symbol. If the name is determined to be invalid, it is ignored, and normal processing continues. In conversational mode, EVAL is then called to examine the operand. Only absolute, null, and indeterminate expressions are considered valid; absolute and indeterminate expressions must be delimited by a blank. A null expression may consist of either a comma or blank. In batch mode, control is returned to the caller after the name field check.

Error Checks:

- The value of the single field must be null, absolute, or indeterminate.
- The single operand field is delimited by a blank.
- The name field is blank or contains a sequence symbol.

TITLE -- TITLE Instruction Scan (CEVTI)

This routine checks the TITLE instruction for correct format. It assumes that the caller has previously called BASCAN to examine the name field of this instruction and that all storage areas affected by that call are unchanged except BSSCAN, which has been set to the address of the first character in the operand. (See Chart BC.)

Entry Point: CEVTI

Calling Sequence: INVOKE ATITLE

Routines Called: BASCAN, DIAG

Exit: Normal

OPERATION: The name field is checked for up to four alphabetic or numeric characters. If found, and this is the first TITLE instruction, the field is saved for later use in card identification. The operand field is checked for a character string with a maximum length of 100 characters. If a character string is found, it is saved for later use in printout of assembly listing. If the character string length exceeds 100 characters, only the first 100 characters are retained for later use. Control is then returned to the caller.

Error Checks:

- The name field contains an invalid symbol.
- The name field of the second and following TITLE instructions is not blank.
- The name field of TITLE exceeds four characters.
- An invalid operand field.
- A truncated value (when operand exceeds 100 characters in length).

USE/DROP -- USING and DROP Instructions Scan (CEVUD)

This routine examines DROP and USING instructions for valid operand fields and

correct formatting. It assumes that the caller has previously called BASCAN to scan the name field of this instruction and that all storage areas affected by that call are unchanged except BSSCAN, which has been set to the address of the first character in the operand field. (See Chart BD.)

Entry Points: CEVUD1, CEVUD2

Calling Sequence: INVOKE AUSING or ADROP

Routines Called: DIAG, EVAL

Exit: Normal

OPERATION: The name field of the statement must be blank or must contain a sequence symbol. In the event of an invalid name field, a warning is issued and the name is ignored. In batch mode a GSM entry is made and control is returned to the caller. Otherwise, the instruction operand is examined. For USING instructions the first operand field is evaluated to see that it is valid; it must be relocatable, absolute, complex, or indeterminate.

The register designations are examined to see that they are absolute or indeterminate expressions. If general register 0 is specified in a USING statement, its treatment will be the same as for any other general register. The user can thus conveniently address page 0 of virtual storage by specifying general register 0 as a base register. However, an element of relocatability is lost. Any area covered by GR 0 is effectively the same as specifying no base register at all, and hence cannot be relocated at execution time. For DROP statements the validity of the register designations is checked. If no operands are present, all registers which have been designated as base registers will be dropped. Registers which are specified must be specified by absolute or indeterminate expressions. The delimiters are checked and must be a comma or blank. A count is kept to make sure that only 16 registers are listed. GSM entries are created for either instruction, and control is returned to the caller.

Error Checks:

- At least one base register designated.
- The number of register designations less than or equal to 16.
- The expression representing the first operand of USING statement absolute, indeterminate, complex, or relocatable.
- The expression designating the register absolute or indeterminate.

- The operand fields delimited by a comma or blank.
- The name field blank or contains a sequence symbol.
- A given register not designated more than once in any single instruction.
- The absolute value designating a register less than or equal to 15.

MACREF -- Macro Reference Processor (CEVRF)

This routine is called when analysis of the current operation mnemonic indicates a macro instruction statement. It is responsible for indicating the presence of the macro instruction in Phase I and for ensuring the presence of the macro definition in Phase IIA. (See Chart BE.)

Entry Points: CEVRF, CEVRF1

Calling Sequence: INVOKE AMCREF
or
L R2,RF011
BR R2

Routines Called:

- Internal - DIAG MACLKT VMGET
DLKM PARAMAC
- External - CEVMLA, CEVMLB

Exit: Normal

OPERATION: Macro instructions are directed to this routine. To provide the type attribute M, this routine first determines whether there was a symbol in the name field of the source line. If so, a transitive item is constructed, if one does not already exist, and this item is flagged. In Phase I, an entry for the GSM chain is constructed to cause the expansion of the macro instruction during Phase IIA. In Phase IIA, the current statement is a nested macro instruction and is to be expanded on the spot.

If the macro definition does not currently exist in assembler working storage, I/O activity is initiated to read the source lines into working storage and chain the lines together. The input source mode of REED is then set to the macro library mode, and REED is directed to the prototype line of the definition. The macro definition switch is set to 1, indicating a definition; control is transferred to the input branch of STAN with an open transfer (not a subroutine linkage); and an indicator is set to denote MACREF mode.

Subsequent statements are procured from library lines previously stored and are interpreted as a macro definition. The occurrence of the MEND statement when MACREF mode is set re-directs control to this module at the point of departure.

PARAMAC is called to perform initialization of the macro level dictionary and set the input source mode of REED to macro definition mode so that, upon return to STAN, subsequent statements will be procured from the macro definition currently referenced.

If the macro definition exists in storage, the process of obtaining the definition from the library is bypassed, and PARAMAC is called directly.

Error Checks:

- Duplicate symbol.
- Error in macro library retrieval.
- Error in library macro definition.

MACDEF -- Macro Definition Processor (CEVDF)

This routine controls the processing of macro definitions. It monitors statements appearing between MACRO and MEND statements, and is responsible for entering the name of the macro in the macro name dictionary. (See Chart BF.)

Entry Point: CEVDF

Calling Sequence: INVOKE AMCDEF
... (error return)
... (normal return)

Routines Called: CATOP MACLKT
COPY MACPUT
DIAG SUBOP
GETOP

Exit: Normal

Error - ISEQ or ICTL within a macro definition or macro redefines an assembler mnemonic.

OPERATION: This routine is called by STAN when it is known that a macro definition is being input (MDS is not zero). At this point it is known also that the operation code of the current line is neither MEND nor MACRO.

If MDS equals 1, this indicates the previous source line was a MACRO statement and therefore the current line is the prototype statement. A macro name item is constructed in the main dictionary using the

contents of the operation code field (maximum eight characters) as the key. In addition, the operation code is looked up in the operation code table and, if a match is found, and the code is a machine operation, a diagnostic is issued warning that a machine mnemonic code has been redefined by a macro definition. The redefinition indicator is turned on in the matching operation code table entry. If the redefined operation is an assembler instruction, the statement is diagnosed as illegal, and all statements except END are treated as commentary. The dictionary item for the macro name is completed by inserting the location of the LOF entry for the prototype line and the location of the prototype line itself. The former is used by MACREF in initializing REED to read the definition when the macro is expanded. The latter is used by PARAMAC to establish a temporary macro level dictionary when the macro is expanded.

After establishing the macro name item, return is made to STAN. Thereafter, incoming statements from STAN pass through this module; the operation codes ISEQ, ICTL, and END are diagnosed as illegal within a macro definition. The LOF entry for an ISEQ or ICTL is deleted, and an error return is made. If the statement is END, the end indicator (ENDIND) is set, and a normal return is made. COPY statements cause the COPY module to be called, which reads in the library element and pushes down the input source switch in REED so subsequent statements originate from the library. Copied statements thus become part of the macro definition and not part of the expansion. COPY statements during expansion are suppressed by REED in the macro definition mode.

Error Checks:

- ISEQ, ICTL or END within macro definition.
- Macro redefines an assembler instruction.

CSCAN -- Constant Scan (CEVCS)

This routine collects and analyzes each subfield of a data definition. It obtains and, when necessary, converts the constant to produce constant items for each operand examined. The attributes of the constant are also evaluated. (See Chart BG.)

Entry Point: CEVCS

Calling Sequence: INVOKE ACSCAN

Routines Called: BASCAN EDEC
DIAG EVAL

Exit: Normal

Input Parameters:

BSSCAN Word pointer to location of first character of expression

Output Parameters:

- R0 Beginning address of constant item; zero if scan mode or unable to process.
- R1 Set to 1 if multiple operands; otherwise, it is zero.
- R2 Set to 1 if a location counter reference is encountered in A, S, Y types; otherwise, it is zero.

OPERATION: This routine is called to process the operands of DC, DS, and DXD statements, by Phase IIB to process literals, and by the literal scan routine in EVAL to obtain the character length of the source text that represents the entire literal.

If the literal scan mode switch is on, a constant item is not generated, and all diagnostics are suppressed except those pertaining to literals.

If the literal scan mode switch is not on, a constant item is formed for the operand, and, when necessary, the constant is generated, and its value appended to the constant item. The length attribute is computed for all data types. When appropriate, the integer and scale attributes are also computed. In the case of multiple constants, these attributes are those of the first constant in the set. A DC statement with a zero duplication factor is treated as if it were a DS statement. No values are attached for DS statements and for address constants.

A call to this routine causes a single operand to be processed; however, the operand field may contain multiple operands. If the delimiter that terminated the operand field was a comma, the caller is notified so that this module may be reentered to process the remaining operands.

Error Checks:

- Zero duplication factor in literal.
- Truncated value.
- Invalid hexadecimal constant.
- Invalid binary constant.
- Floating point characteristic out of bounds.

- All precision lost during scaling.
- Invalid decimal constant.
- Improper operand for V-type address constant.
- Improper operand for R-type address constant.
- Invalid delimiter.
- Invalid type subfield.
- Value of length modifier invalid for type of constant.
- Scale modifier not permitted for type of constant.
- Exponent modifier not permitted for type of constant.
- Exponent modifier out of range.
- Scale modifier out of range.
- Multiple constants not permitted for type of constant.
- Data omitted from DC operand.
- Invalid operand field.
- Improper operand for Q-type address constant.

SSCAN -- String Substitution Scan (CEVSS)

This routine performs string substitution of variable symbols and symbolic parameters appearing in a source statement. (See Chart BH.)

Entry Points: CEVSS, SSCAN2

Calling Sequence: INVOKE ADSCAN
 ... (error return)
 ... (normal return)

Routines Called: DIAG DLKT VMGET
 DLKM EVAL

Exit: Normal
 Error - Invalid parameter or substring notation

OPERATION: Translate and test instructions are used to isolate variable symbols in the statement to be scanned. The variable symbol isolated by this method is then looked up in either the macro level dictionary or main dictionary, whichever is pertinent. The dictionary item for the symbol yields either a character string or other value, which is then substituted for the symbol in a reconstructed version of the statement.

Special rules govern substitution in the operand field of SETB and AIF instructions. SETB symbols represent Boolean values which will not be apparent to EVAL if replaced by the corresponding character string value. Accordingly, SETB symbols appearing within quotation marks are always replaced. Outside quotes, the character string is substituted only when concatenation to the surrounding character string is indicated. Substitution is not performed if the SETB symbol, including subscripts, is delimited on both the left and the right by an arithmetic operator (+, -, /, or *), a parenthesis, or a blank. The character string value is concatenated if the SETB symbol is delimited on either side by a character other than those mentioned above.

In the operand field of SETC instructions, and for character values found in logical expressions, two or more character expressions are concatenated into a single string when required. Substring notation is interpreted and the selected substring is substituted when such notation is present.

Special processing is performed when attribute notation is encountered. If the count or number attributes of a symbolic parameter or the type attribute of a symbolic parameter whose argument is not a symbol are requested, the respective integral or character string value of the attribute is substituted for the attribute notation. If the argument is a symbol, the parameter is replaced by the argument, and the attribute notation is left intact; its value is obtained subsequently. This difference in processing arises because N' and K' are attributes of the symbolic parameter itself, whereas T is an attribute of the argument if the argument is a symbol, but is otherwise an attribute of the parameter.

Similar treatment is also performed for the substitution of system variable symbols and their attributes.

EVAL -- Expression Evaluator (CEVEV)

This routine evaluates an arithmetic or logical expression designated by the calling module, and returns with the value and type of the expression. (See Chart BI.)

Entry Point: CEVEV

Calling Sequence: INVOKE AEVAL

Routines Called: BASCAN DLKT EDEC
 CSCAN DLPM EHEX
 DIAG EBIN PSCAN
 DLKM ECHAR

Exit: Normal
 Error - Unbalanced parentheses or
 invalid parameters

Input Parameters:

BSSCAN Word pointer to location of
 first character of expression

FAL 1-byte flag with value of 0
 (arithmetical expression) or 1
 (logical expression)

Output Parameters:

R0 Value of the expression, if abso-
 lute value; absolute part if
 relocatable.

R1 Location of RLD string, if relocat-
 able; otherwise, no information.
 For a description of the RLD string,
 see "Comments," below.

R2 Length of RLD string (in bytes) if
 relocatable; otherwise, no
 information.

FEX 1-byte exit flag; indicates type of
 expression evaluated and set as
 follows:

Expression Type	Hex Bit Value Set
Absolute Arithmetic	00
Absolute Boolean	01
Relocatable	02
Literal	04
Indeterminate	08
Error in Syntax	10
Null; first character was end-of-expression delimiter	20
Complex	40

OPERATION: Following is a description of
 the operation performed for each function
 of the routine.

Scanning Techniques: BASCAN is employed to
 scan the expression. The normal output of
 this routine is either a term followed by
 an operator (delimiter) or an operator alone
 (null term).

The right delimiter serves as an inter-
 pretive key to the expression. In an arith-
 metical expression, a comma or space indi-
 cates the end of the expression. A left or
 right parenthesis may also indicate the end
 of the expression, subject to the following
 conditions:

- A left parenthesis following a preced-
 ing left parenthesis or operator is
 considered to be algebraic. A left
 parenthesis following a term is consi-
 dered the end of the expression, unless

the term itself requires subscript
 notation. In the latter case, appro-
 priate indicators are placed in tables
 internal to this routine and a recur-
 sive evaluation of the expression for
 the subscript is started.

- For a logical expression, an unbalanced
 right parenthesis indicates the end of
 the expression. Spaces serve to separ-
 ate the logical and relational opera-
 tors but do not delimit the expression
 as a whole. The calling routine indi-
 cates by setting a switch whether this
 routine is to operate in the arithmetic
 or logical mode.
- In the logical mode, a symbol occurring
 where an operator is expected causes
 this routine to test the spelling of
 the symbol against that of the logical
 and relational operators. If a match
 is found, the symbol is treated as an
 operator.

Interpretation of Terms: BASCAN returns to
 this routine with a character substring
 consisting of a term followed by its right
 delimiter, or with a right delimiter only
 (a null term). It also provides an indica-
 tor to tell whether a term is present, and
 if so, the type of term. This indicator is
 tested for one of the following seven poss-
 ible outcomes:

- There is no term (a null, or operator,
 or right delimiter only): control is
 transferred to the lone operator rou-
 tine, which is described in a later
 section.
- The term is a literal: in Phase III of
 the assembler, the normal procedure is
 to (1) create a hash number from the
 first eight characters of text (exclud-
 ing =) and inserting blanks in the
 lower order bytes, if necessary, (2)
 locate the literal in the dictionary,
 and (3) create an RLD output list and
 classify the expression as relocatable.
 In earlier phases, the literal is mere-
 ly scanned over, and the expression is
 classified as indeterminate.
- The term is the location counter *:
 the location counter is relocatable, so
 unless the expression contains a term
 that can be paired with it, the result-
 ing expression is relocatable also.
 The location counter indication is
 placed in an RLD string; its format is
 the same as a dictionary item.
- The term is a symbol name: it may be
 absolute, relocatable, undefined, or
 complex. The normal procedure is to
 create a hash number from the symbol
 name, look it up in the dictionary, and

get the value from the value field in the dictionary. The exceptions to this procedure are: (1) The symbol name is not in the dictionary -- the term is indeterminate. A code will be set and an exit to the calling routine will be made. (2) The symbol was found, but the value was not there, meaning that the term is relocatable. The term will be put into the RLD string. If the associated operator is not ADD or SUBTRACT, there is an error, and an error code will be set and control will be returned to the calling routine. (3) The term is an external reference; an indicator code will be set and an exit will be made.

Cross-reference items are prepared by a subordinate routine for each symbol encountered when the cross-reference mode indicator is set.

- The term is an attribute of a symbol: the normal procedure is to create a hash number from the symbol name, find the symbol name in the dictionary, get the specified attribute from the attribute field in the dictionary, and use this as the value of the term. The exceptional procedure occurs when the symbol name is not in the dictionary; an error code will be set, and an exit will be made.
- The term is self-defining: an indicator from BASCAN indicates the type of self-defining term. The term will be evaluated.
- The term is absolute Boolean; a diagnostic will be issued and an exit will be made.

Interpretation of Parentheses: A left parenthesis as a right delimiter has two interpretations:

- It is the end of the expression and introduces a subfield.
- The term is a subscripted variable symbol, and the item being introduced by the parenthesis is an expression that will identify the correct value from the group of values associated with this term.

When the term is found in the dictionary, the maximum subscript value field can be tested to determine which interpretation is correct. If the field is zero, there is no subscript; if it is not zero, a subscript must be present. If the term is nonsubscripted, the left parenthesis terminates the current expression. Evaluation of the expression is completed

and control is returned to the calling routine. If the parenthesis level counter indicates that the expression is not finished, an error message is given.

If the term is subscripted, a flag is placed in the waiting stack (see "Order of Scan"), while the expression inside the parenthesis is evaluated. If one of the terms inside the parentheses also has a subscript, another flag is placed in the waiting stack and that expression is evaluated. Flagging and stacking of subscript expressions in the stack can be carried on indefinitely. When a subscript expression is finally evaluated, the proper value for the term can be found from the dictionary. At this point the flag is removed from the waiting stack, and evaluation of the previous expression can be resumed. A left parenthesis that follows a right parenthesis, (, indicates an end of the expression. The left parenthesis is presumed to introduce a subfield that has followed a subscripted set symbol.

Determination of Order: The order in which the operations in an expression are performed is controlled by the level of parentheses and by the hierarchy of the operators. Operators of the same hierarchical and parenthetical level are performed from left to right, as they are encountered by the scanner.

Since the parenthetical and hierarchical levels of the operators may dictate a sequence of operations that is different from the order in which they are encountered by the scanner, a pushdown (last in, first out) table is created for holding those operations that have been scanned, but whose execution must be delayed. Entries in this table consist of a term, an operator, and a parenthesis-level indicator. There is also an index associated with the table that identifies the most recent entry in the table.

In performing required operations, the hierarchical level of the current operator is compared with that of the next operator; if the level of the next operator is higher, the current operator, and its terms, are placed in the waiting stack. Then the next term and operator become the current term and operator, and a new next term and operator are obtained from the scanner.

Whenever the new operator has a lower hierarchy than the current one, the current operation is performed. The next term and operator are then retrieved from the waiting stack, whose index is then reduced. Parentheses override the hierarchy of

operators. When a left parenthesis is encountered in the position of an operator, the current operator and term are placed in the waiting stack. The parenthesis level counter is increased by 1 and placed in the stack also. The evaluation is continued as above until a right parenthesis is encountered. At this time the operation for the current level of parenthesis in the waiting stack is performed. When these are completed, the parenthesis level counter is reduced by 1.

Logical Expressions: The calling routine will set a flag when the expression that is presented to this module is expected to contain logical or relational operators. This will occur when the statement is a SETB or AIF assembler instruction. This flag signals the coding that is normally bypassed to be executed, and changes the way in which certain delimiters are interpreted. An unbalanced right parenthesis or end-of-statement control will now indicate end of expression, and these methods are the only legal ways to end the expression. When a symbol occurs in a position within the expression normally occupied by an operator, the symbol is compared with a table of logical and relational operators. If it matches one of them, a one-byte code is used to represent the operator. If no match occurs, the symbol is assumed to be a term, and the previous item is compared with the unary operator NOT. If this is a match, the unary NOT flag is set; if not, DIAG is called to indicate that two terms have been written without an operator, and an exit is made. The hierarchy of the logical and relational operators is meshed with the arithmetic operators and evaluation of the expression proceeds exactly as it does for an all-arithmetic expression, except that an expanded list of operators is accepted, and logical terms can only be combined with other logical terms. The value of an expression containing a logical or relational operator is a logical 1 or 0, depending on whether the expression is TRUE or FALSE, respectively.

Reduction of Relocatable Expressions: If there is more than one relocatable term in an expression, it is possible that they may be paired off in a way that cancels the relocatable aspect of the terms and produces an absolute expression. Every effort is made to do this. For instance, if terms A and B are relocatable, and if they occur in the same control section, the expression $(A+5-B)*2$ is absolute. $(A+B)*2$ is not, because the sum of A and B is not fixed, and the relocatable aspect will not cancel. When relocatable terms are encountered in the expression, the location in the dictionary of the term, the operator, and the parenthesis level are placed in an RLD string. (If the operator is not ADD or SUBTRACT, DIAG is called and an exit is made.)

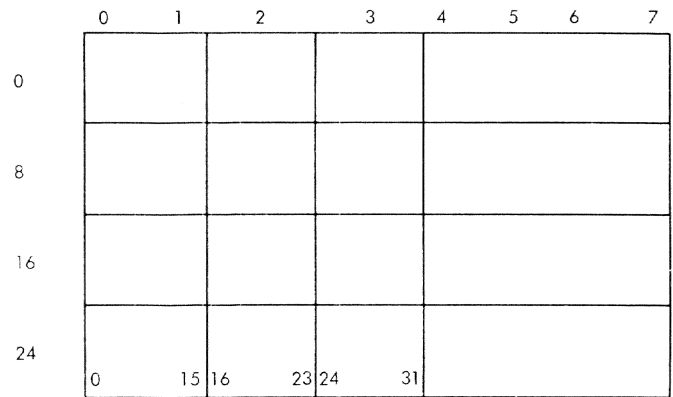


Figure 17. Waiting stack format

At each parenthesis level an attempt is made to pair off all relocatable terms in the RLD string. If terms are in the same control section and have opposite signs, they may be paired. If the resulting value of a given parenthesis level is absolute, an add, subtract, multiply, or divide operation may be performed on it. If it is relocatable, only an add or subtract operation may be performed, and any other operation will cause an error message and an exit. The final value of an expression that contains relocatable terms may be relocatable or absolute, depending on whether all the relocatable terms can be paired off. If the final value is relocatable, the output will consist of an absolute part plus an RLD string containing the unpaired relocatable terms and their operators and the location and number of entries in the RLD string.

Order of Scan: The sequence in which the operations in an expression are performed is determined by parentheses and by the hierarchy of the operator. A table called the waiting stack (Figure 17) is created to hold operations that cannot be performed at the time they are scanned. Scanning the expression is done from left to right, and is performed by BASCAN. On each calling of this module, it gets the next string, which usually consists of a term followed by a delimiter (or operator).

The format of the waiting stack is shown in Figure 17. Each entry is 8 bytes long as follows:

<u>Byte</u>	<u>Content</u>	
0-1	Parenthesis level (number of left parentheses encountered minus number of right parentheses encountered)	
2	Operator	
	<u>Arithmetic</u>	<u>Hex value</u>
	Addition (+)	F4
	Subtraction (-)	F6

Multiplication (*)	F8
Division (/)	FA

Logical

OR	08
AND	0A
NOT	0C
LE	C0
GE	A0
EQ	80
NE	70
LT	40
GT	20

Byte	Content
3	Flags
	<u>Bit</u>
	0 K attribute expression
	1 SETB statement
	2 Relocatable expression
	3 Subscripted expression
	4 N attribute expression
	5 New parenthesis level
	6 Logical operator
	7 Arithmetic operator
4-7	Value of the expression following the operator.

Hierarchy of Operators: The hierarchy of the next operator is compared with that of the current operator (which is initialized to +). If the next operator is equal or lower than the current operator in hierarchy, the operation between the current value (current term), current operator, and next term is performed. This result becomes the current value, the next operator becomes the current operator, and a new string is obtained.

If the next operator is higher than the current operator in hierarchy, the waiting stack pointer is stepped by 1 entry (8 bytes); and the current value, current operator, and current parenthesis level counter (CPLC) are placed in the stack. The next operator and next term become the current value (term) and current operator and the next string is obtained.

When the delimiter is an end of expression indicator -- blank or comma, for instance -- it is treated as an operator of lowest hierarchy. The operation involving the current value, current operator, and next term is performed. The parenthesis level of the entry in the stack designated by the waiting stack pointer is compared with the CPLC. If equal, the operation between the waiting term, waiting operator, and current value is performed. The stack pointer is stepped back by one entry, and another parenthesis level test is made. When the parenthesis level in the stack is not equal to the CPLC, there are no more items in the stack at this parenthesis level, and the current value is the final value for that parenthesis level.

Hierarchy of Parentheses : Upon calling the scan, a null string, consisting of a delimiter only, with no term, may be returned. When this delimiter is a left parenthesis, the CPLC is stepped by +1; the stack pointer is stepped to the next entry; the current value, current operator, and CPLC are put in the stack; zero replaces the current value; and ADD replaces the current operator. Then BASCAN is summoned to fetch the next string.

When the delimiter of a term is a right parenthesis, this is treated as an operator of low hierarchy, and the current operation is performed, and all the operations in the stack at the current parenthesis level are performed. Then the CPLC is stepped back by 1 and scanning continues.

If the next string is a lone right parenthesis (without a term), all the operations in the stack at the current parenthesis level are performed, the CPLC is stepped back by 1, and scanning is resumed.

If, following a right parenthesis, the next string is a null string consisting of an arithmetic operator, this becomes the current operator and the next string is obtained.

Retrieval of Subscripted Values: Two conditions must be satisfied for a term to have a subscript:

- The right delimiter of the term must be a left parenthesis.
- The term must be a variable symbol defined in the dictionary as a subscripted set symbol parameter, or the system symbol SYSLIST.

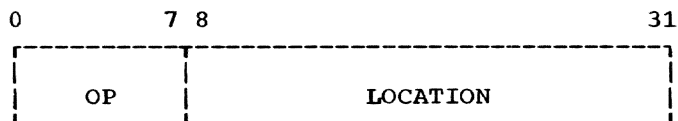
If the first condition is met and the second is not, the left parenthesis is an expression delimiter (which is also introducing a subfield expression). If only the second condition is met, a diagnostic is issued, and a subscript value of 1 is assumed. These tests are made while the item is still in the position of being the next string. If both conditions for a subscripted term are met, the stack pointer is stepped by one entry and the current value and current operator are put in the stack; then the current value is set to zero and the current operator is set to +. The CPLC is stepped by +1, the stack pointer is stepped by one entry, and the dictionary location of the term is placed in the stack. A subscript flag is also put in the stack. The operator of this term has not been scanned yet and will be placed in the stack later. A flag is set which will indicate that any relocatable term that is now encountered is an error. This expression

is now evaluated in the usual way by calling BASCAN to get another next string. Another subscripted term may occur within this expression. It will be handled as described above.

When the delimiter of the subscript expression is encountered, the items waiting in the stack are pulled out and processed as usual, until the last one in the stack is encountered. This last item is identifiable in two ways: (1) it is the last item of the current parenthesis level (indicated by CPLC), and (2) the dictionary location of the term will be present in the stack (i.e., this field is nonzero). The type field in the dictionary is tested, and the proper value for the nth subscript is taken from the dictionary.

For SET symbols and %SYSLIST, the subscripted value is retrieved directly. For parameters, a subordinate module PSCAN is called. PSCAN examines the argument string of the nth sublist operand for validity. The argument string must be interpretable as a self-defining term for the subscript expression to be valid. If the argument is a self-defining term, its binary value is obtained by PSCAN and given to this module upon return. The accumulated value of the previous expression is in the stack and can be reached by stepping the stack pointer back by one entry and reducing the CPLC by 1. The value of the current term comes from the dictionary (via the subscript) and the operator for it comes from BASCAN.

Comments: The RLD string consists of entries of the following format:



An entry of this type is completed for each relocatable symbol in an expression. All entries for symbols in a given expression are combined to form the RLD string.

OP represents the operation associated with a relocatable symbol; this is, by definition, limited to either subtraction or addition. LOCATION is a pointer to the relocatable value item for the symbol and is the means by which the section number and displacement of a relocatable symbol can be procured. There is no limit placed upon the number of terms that may be contained in an expression.

Error Checks:

- Undefined symbols
- Invalid expression type for field

- Invalid attributes
- Multiple literals
- Improper formation of logical expressions
- Invalid self-defining terms
- Unbalanced parentheses
- Consecutive terms and operators
- Number of terms exceeding OS compatibility limits
- Multiplication or division of relocatable terms
- Invalid subscript values
- Invalid variable symbols and parameters
- Missing subscripts
- Arithmetic overflow during evaluation

PSCAN -- Parameter Item Analyzer (CEVPS)

This routine determines whether the argument string for a parameter consists of a self-defining term. (See Chart BJ.)

Entry Point: CEVPS

Calling Sequence:

```

INVOKE  APSCAN
...     Other return (4 bytes)
...     Self-defining term return

```

Input Parameters:

```

R1      Location of parameter item
R2      Value of subscript to apply, if any

```

Routines Called: EBIN EDEC
ECHAR EHEX

Exit: Normal

Output Parameters:

```

R1      Value of self-defining term (32 bits)

```

OPERATION: This routine examines the argument character string contained in a parameter dictionary item. If the character string exhibits the characteristics of a self-defining term, the appropriate conversion subroutine (subordinate to EVAL) is called to convert the term to its proper value, which is then returned to EVAL as the value of the parameter symbol. If the argument string does not appear to be a self-defining term, an alternate exit is

taken; EVAL then considers the expression containing the parameter to be unevaluable.

Comment: The argument strings for the majority of parameter symbols are substituted into the source statement by SSCAN, prior to assembly of the statement. However, when the parameter which SSCAN is substituting is subscripted, EVAL is called to provide the value of the subscript. The assembler language permits parameters to appear within the subscript; thus EVAL may encounter a parameter symbol which has not yet been substituted by SSCAN. In this case, the parameter is legitimate if its argument string can be interpreted as a self-defining term. This routine makes this determination and provides the value of the term for EVAL. EVAL allows for nested subscripts.

EBIN -- Binary Self-Defining Term Generator (CEVGB)

This routine converts a character string of zeros and ones into a binary integer. (See Chart BK.)

Entry Point: CEVGB

Calling Sequence: INVOKE AEBIN
... Return

Input Parameters:

- R1 Length of character string to be converted, in bytes.
- R3 Virtual storage address of first character of string.

Routines Called: DIAG

Exit: Normal

Output Parameter: R6 Result

OPERATION: The length of the character string is tested against 32. If it exceeds 32, a warning message is printed. Then the bit string is truncated on the left, and the rightmost 32 bits are accepted into the character string. BASCAN has already checked for any character in the string which is not zero or not one, and a warning message was printed. These characters are transmitted to this module unaltered, and this module interprets characters with odd codes as 1 and even codes as 0. Thus, the letter I is interpreted as 1 and letter O as 0.

EDEC -- Decimal Self-Defining Term Generator (CEVGD)

This routine takes a character string of numeric digits and converts it into a binary integer. (See Chart BK.)

Entry Point: CEVGD

Calling Sequence: INVOKE AEDEC
... Return

Input Parameters:

- R1 Length of character string to be converted, in bytes.
- R3 Virtual storage address of first character of string.

Routines Called: DIAG

Exit: Normal

Output Parameters: R6 Result

OPERATION: If the character string exceeds 10 digits, if a character which is not of the decimal set is encountered, or if the value of the number exceeds $2^{31}-1$, a warning message is printed. Then the routine exits with $2^{31}-1$ as the result. Otherwise, the decimal value is converted to its binary equivalent, and exit is made.

EHEX -- Hexadecimal Self-Defining Term Generator (CEVGH)

This routine converts a string of hexadecimal characters into its binary equivalent. (See Chart BL.)

Entry Point: CEVGH

Calling Sequence: INVOKE AEHEX
... Return

Input Parameters:

- R1 Length of character string to be converted, in bytes.
- R3 Virtual storage address of first character of string.

Routines Called: DIAG

Exit: Normal

Output Parameter: R6 Result

OPERATION: The length of the character set is tested against 8. If it exceeds 8, a diagnostic message is printed. If it contains fewer than 8 characters, the resultant word is zero filled on the left to make up the difference. Then the bit

string is truncated on the left, and the rightmost 32 bits are accepted into the bit string and placed in the output word. If a character that is not of the hexadecimal set is encountered, a warning diagnostic message is printed. The lower four bits of the 8-bit character (i.e., the numeric portion) are taken to form the hexadecimal digit.

ECHAR -- Character Self-Defining Term Generator (CEVGC)

This routine takes a character string and puts it in the format of a character self-defining term. (See Chart BL.)

Entry Point: CEVGC

Calling Sequence: INVOKE AECHAR
... Return

Input Parameters:

- R1 Length of character string to be converted, in bytes.
- R3 Virtual storage address of first character of string.

Routines called: DIAG

Exit: Normal

Output Parameter: R6 Result

OPERATION: This routine takes a string of characters and formats them into one full word. If the input character string contains fewer than four characters, the word is zero filled on the left, and the characters are right aligned in the word. If the input character string exceeds four characters, a diagnostic message is printed. Then the character string is truncated on the left, and the rightmost four characters in the string are assembled in the output.

SLIT -- Scan for Literal Operand (CEVSL)

This routine scans the operand of a machine instruction to determine if a literal operand is present. If a literal is found, its location is added to the current logical order file entry. (See Chart BM.)

Entry Point: CEVSL

Calling Sequence: INVOKE ASLIT

Routines Called: None

Exit: Normal

OPERATION: This routine is called in batch assemblies. It uses the logical order file entry to locate the operand field of the current statement, then performs a translate and test instruction to locate a literal operand (denoted by the character =). If a literal is found, the location of the = in the source statement is added to the current logical order file entry.

DLPM -- Dictionary Lookup and Put (CEVLP)

This routine finds the location of a given symbol in the main dictionary, and creates a skeletal dictionary item for the symbol if it has not been entered previously. (See Chart BN.)

Entry Point: CEVLP

Calling Sequence: INVOKE ADLPM

Input Parameters:

- R0 Location of the first byte of the name to be found or put in the main dictionary.
- R1 Length of the name (in bytes) specified in R0.

Routines called: None

Exit: Normal

Output Parameters:

- R2 Location in the main dictionary where the given name was found or placed.
- R3 Zero = name was placed in dictionary.
Nonzero = name was already in dictionary.

The condition code also reflects this outcome:

- 0 = name was placed in dictionary.
- 1 or 2 = name was already in dictionary.

- R6 Address of hash table value for the symbol in question.

OPERATION: The main dictionary is searched for the given name. V-type external definition items are invisible to this search. If the name is found and is not in a transitive item, this module returns to the calling routine with the location of the name in the main dictionary in R2. R3 will be nonzero, and the condition code will be 1 or 2.

If the name is not found or is found in a transitive item, a skeletal dictionary entry is created in the next available location in working segment 2 (via AWORK2). Specifically, the name is placed in WORK2 through WORK2+7, the location of the next hash synonym (relative to the base of working segment 2) is placed in WORK2+9 through WORK2+11, and WORK2+8 is cleared. AWORK2 is not updated; the caller must update this value when he completes the entry. The location of the dictionary item is passed back to the caller, with an indication as to whether the symbol previously occurred in the name field of a macro reference.

DEFSYM -- Define Location Symbol (CEVSY)

This routine constructs and enters into the main dictionary a relocatable value item representing the name field symbol (if present) of the current source statement. (See Chart BO.)

Entry Point: CEVSY

Calling Sequence: INVOKE ADFSVM

Routines Called: DIAG, DLPM

Exit: Normal

OPERATION: If the name field is blank or contains a sequence symbol or BASCAN indicated an error, control is immediately returned to the caller. Any other type of entry in the name field is diagnosed as an error and ignored; an exit to the caller occurs.

A valid symbol in the name field causes the main dictionary to be searched for that symbol. If the same symbol was defined previously, an error has occurred. A diagnostic is issued, the first definition is honored, and control is returned to the caller. If the symbol had not been defined, a dictionary item is created for that symbol. The type of dictionary item made and the amount of information included in the item are a function of the instruction type being processed. At this point control is returned to the caller.

DIAG -- Diagnostic Message Processor (CEVDX)

This routine inserts specified variable information into a specified diagnostic message and disposes of the message. (See Chart BP.)

Entry Point: CEVDX

Calling Sequence: INVOKE ADIAG
... Return

Input Parameters:

- R0 Location of source (nonstandard) variable information, if any; or location of operand text of MNOTE instruction (character following the opening quote); if standard variable information is used.
- R1 Lengths in bytes of source variable information. If required by the message, R1 contains an index into the standard variable table. If MNOTE, R1 contains the length of the operand.
- R2 Location of diagnostic entry relative to base of text locator table. If sign bit is on, indicates MNOTE parameters; for MNOTE, bits 24-31 contain the severity code of the MNOTE instruction.

Routines Called:

- Internal - None
- External - CFADC1 Accept diagnostic entry of language processor control

Exit: Normal

OPERATION: This routine is supplied a message number which it uses as an index into a text locator table starting from location CEVDX3. Entries in this table are one word each with the format shown in Figure 18. A record is kept in SEVCO of the highest severity code encountered.

Variable information for messages is limited to eight bytes at the beginning of the message. The text of the standard variable information is kept in a table of doubleword entries (Table 4). An index into this table is supplied with the message number whenever standard variable information is to be contributed to the message.

If the variable information is to come from the source statement, the length and location of the source characters are supplied.

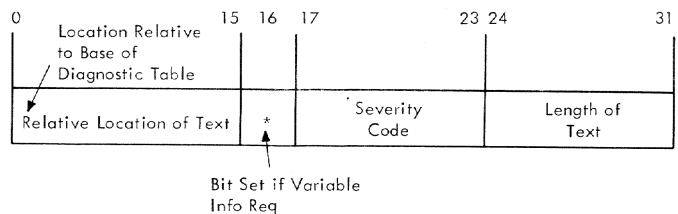


Figure 18. Diagnostic text locator entry format

Table 4. Standard variable information table

1 - R1	17 - OPERAND
2 - R2	18 - L
3 - R3	19 - M1
4 - S1	20 - COMMAND
5 - S2	21 - DATA ADR
6 - D1	22 - FLAG FLD
7 - D2	23 - COUNT
8 - L1	24 - EXPONENT
9 - L2	25 - SCALE
10 - I	26 - BINARY
11 - I2	27 - DECIMAL
12 - B1	28 - FIX-PT
13 - B2	29 - FLOAT-PT
14 - X2	30 - HEX
15 - NAME	31 - TYPE
16 - OPERAT'N	32 - SUBSTRNG

When the assembly is in conversational mode, the text of the message desired is constructed, and the entry in LPC which accepts diagnostic messages is entered (CFADC1). Messages are classified as global and local. After typing a local message LPC unlocks the terminal keyboard for user reaction. If the message is global, LPC does not unlock the keyboard. In the assembler the global mode is not a function of the content of a message but of the time when the message is produced. That is, all messages generated by Phase IIA are effectively global. This condition is indicated by a mode indicator which is interrogated before calling LPC.

If the assembly is in batch mode, the information supplied in its calling sequence is transcribed to a logical order file entry. The listing routine in Phase III will then construct and format the message during preparation of the machine language listing.

DLKT -- Lookup Temporary Dictionary Item (CEVTK)

This routine finds the location of a given symbol in the temporary dictionary for the current macro level. (See Chart BQ.)

Entry Point: CEVTK

Calling Sequence: INVOKE ADLKT

Input Parameters:

- R0 Location of the first byte of the name for which this module is to look.
- R1 Length of the name (in bytes) being sought.

Routines Called: None

Exit: Normal

Output Parameters:

- R2 Location of name, if found. If not found, R2 contains 0.

The condition code also reflects this outcome:

- 0 = not found
- 1 or 2 = found

OPERATION: The temporary dictionary designated by ATSH is searched for the given name. If the name is found, return is made to the calling module with the location of the name in R2. If the name is not found, return is made with zero in R2.

DPUT -- Put Item in Temporary Dictionary (CEVTP)

This routine creates a skeletal item for a given symbol in the temporary dictionary for the current macro level. It assumes the symbol being put into the dictionary has been put in storage area W, left-justified and blank filled on the right. (See Chart BQ.)

Entry Point: CEVTP

Calling Sequence: INVOKE ADPUT

Routines Called: None

Exit: Normal

Output Parameter:

- R2 Location where the new temporary dictionary item was placed. This is the same as AWORK1.

OPERATION: The dictionary is not checked to see if the name has already been placed there. It is assumed that the caller has done this, and this routine is called by the not found path resulting from testing the output from DLKT. It is assumed that DLKT (or some other routine) has set the 12-byte work area W. A skeleton dictionary item is created and placed in the next available location in working segment 1 via AWORK1. Specifically, the name is placed in WORK1 through WORK1+7, the location of the next hash synonym (relative to the base of working segment 1) is placed in WORK1+9 through WORK1+11, and WORK1+8 is cleared. AWORK1 is not updated; the caller must update this value when he fills in the skeleton of the entry.

MACLKT -- Macro Name Dictionary Lookup (CEVLM)

This routine searches the macro name dictionary for a given name. (See Chart BR.)

Entry Point: CEVLM

Calling Sequence: INVOKE ACEVLM

Routines Called: None

Exit: Normal

Output Parameter:

R2 Location of macro name item if the name was found, otherwise zero.

Condition code also reflects these conditions:

0 = not found
2 = found

OPERATION: The macro name dictionary is searched for a name given in location OP. If the name is found, the location of the dictionary item is placed in R2, and the condition code is set to 2. If the name cannot be found, both R2 and the condition code are set to 0. Control is returned to the caller.

MACPUT -- Macro Name Dictionary Put (CEVTM)

This routine inserts an item in the macro name dictionary and the macro name's hash number in the macro name hash table. (See Chart BR.)

Entry Point: CEVTM

Calling Sequence: INVOKE ACEVTM

Routines Called: None

Exit: Normal

Output Parameter:

R2 Location of macro name dictionary item.

OPERATION: This routine assumes MACLKT has been called and:

- The hash table entry has been placed in the work area (W+8).
- OP contains the macro name.

A skeletal item is created in working segment 2, the available core address is updated, and control is returned to the caller.

DLKM -- Main Dictionary Lookup (CEVKM)

This routine searches for a given symbol in the main dictionary and creates a skeletal dictionary item for the symbol if it has not been previously entered. (See Chart BS.)

Entry Point: CEVKM

Calling Sequence: INVOKE ADLKM

Input Parameters:

- R0 Location of the first byte of the name for which this processor is to look.
- R1 Length of the name (in bytes) being sought.

Routines Called: None

Exit: Normal

Exit Parameters:

R2 Location of name if found. If not found, R2 contains 0.

The condition code also reflects this outcome:

0 = not found
1 or 2 = found

OPERATION: The main dictionary is searched for the given name. If the name is found, return is made to the calling module with the location of the name in R2. If the name is not found, return is made with zero in R2.

INTRODUCTION

During Phase IIA, the processing of macro instruction statements begun in Phase I is completed, and system macro definitions are retrieved from the library.

Macro instructions are processed during Phase IIA without reference to other statements in the assembly; therefore, certain supplementary information must be maintained. This supplementary information is obtained from reprocessing all GBL declarations, global SET instructions, section name changes, and PRINT instructions and combining them with macro instructions in the proper order.

Macro statement generation is accomplished by substituting the character string values of the current arguments for the corresponding parameters in the definition. The macro definition statements remain in the sequenced source statement area of assembler virtual storage. Source statements generated by macro instructions are also retained in assembler virtual storage; they do not become part of the set of sequenced statements.

Each new symbolic statement is processed and assembled as if it had been part of the user's original source program. Most of the processing modules used during Phase IIA are the same as those used during Phase I. The Phase IIA control module determines the order and origin of source statements.

All macro instructions have been expanded before exiting from Phase IIA; the logical order of the assembly includes source statements for all generated lines.

Figure 19 illustrates the module relationships in Phase IIA. All the relationships between STAN and the downward associated modules are as shown in Figure 16, except for MACREF. Routine relationships for MACREF that are unique to Phase IIA processing are as shown in Figure 19. For a detailed account of the interaction between STAN and MACREF, see MACREF module description.

PARAMAC constructs a temporary (or macro level) dictionary for each user level and each inner macro instruction.

MACREF, under the control of Phase IIA, searches a macro library index and retrieves model statements from the associated

library. Control is then returned to STAN to process the model statements.

Table 5 is a decision table listing the criteria for entering those routines unique to Phase IIA.

CONVERSATIONAL CONTROL

If the assembly mode is conversational, the transitive item chain in the main dictionary is examined at the end of Phase IIA for undefined symbols, appropriate diagnostic messages are passed to the LPC, and return is made to the initial LPC call. The conversational user may employ other facilities of the system to stop, correct, or continue the assembly.

ROUTINESPHASE IIA -- Phase IIA Control (CEVPB)

This routine controls the expansion of macro instructions. As a corollary to this processing, it reevaluates statements that affect global variable symbols and maintains a history of control section changes. Before concluding, global diagnostic messages are presented to the conversational user, and the language processor control is called to determine whether to continue the assembly. (See Chart BT.)

Entry Point: CEVPBX

Calling Sequence: L R15,ACEVPB
BR R15

Routines Called:

- Unique to Phase II -- PARAMAC (via STAN and MACREF)
- Common with Phase I --
STAN (plus nested modules called)
BASCAN
SETX
GBLx/LCLx
DIAG

Exit: Normal - To AC(CEVAC)
To External Routines - LPC reentry to solicit continuation information (QUERY) via assembler control module (AC).

R14 Return point in LPC.
R15 0 = normal return code.

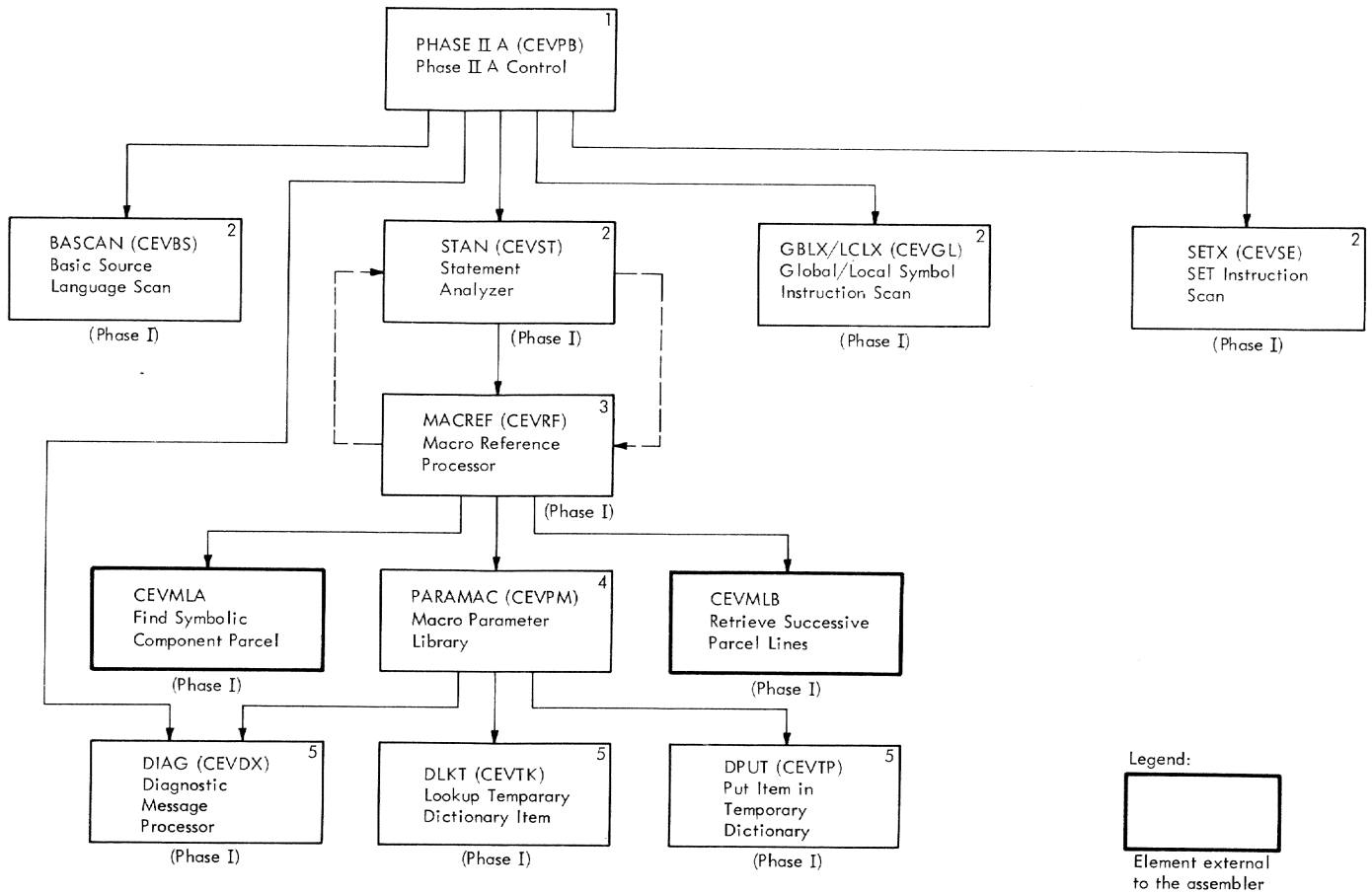


Figure 19. Phase IIA routine relationships

Table 5. Phase IIA decision table (part 1 of 2)

Routine: Phase IIA Control Level: 1			
Routine	Purpose	Called Routines	Calling Conditions
PHASE IIA (CEVPB)	Expands macro instructions.	STAN -- CEVST (Phase I)	Macro instruction encountered in GSM chain.
		DIAG -- CEVDX (Phase I)	D5 Undefined symbol
		BASCAN -- CEVBS (Phase I)	GBLA, GBLB, GBLC, or global SETA, SETB, or SETC statement encountered.
		GBLX/LCLX -- CEVGL (Phase I)	GBLA, GBLB, or GBLC statement encountered.
		SETX -- CEVSE (Phase I)	Global SETA, SETB, or SETC statement encountered.

Table 5. Phase IIA decision table (part 2 of 2)

Routine: Macro Parameter Processor		Level: 4	
Routine	Purpose	Called Routines	Calling Conditions
PARAMAC (CEVPM)	Creates temporary symbol dictionary items for each outer and inner macro instruction level.	DLKT -- CEVTK (Phase 1)	<ol style="list-style-type: none"> One of the following encountered: symbolic prototype label, macro instruction keyword, prototype positional operand, prototype keyword operand. Always called to create the hash table entry for &SYSNDX, &SYSECT, &SYSPSCT, &SYSSTYP, &SYSLIST.
		DPUT -- CEVTP (Phase 1)	<ol style="list-style-type: none"> One of the following encountered: symbolic prototype label, macro instruction keyword, prototype positional operand, prototype keyword operand. Always called to put the following in the TSD: &SYSNDX, &SYSECT, &SYSPSCT, &SYSSTYP, &SYSLIST.
		DIAG -- CEVDX (Phase 1)	D88 Invalid symbolic parameter D100 Invalid keyword D101 Positional operand follows keyword operands D107 Character string accumulation exceeds 255 D115 Invalid operand D117 Nested keyword notation D122 Multiple keyword operand D125 Multiple prototype positional operand with same name D126 Undeclared keyword operand D127 Illegal use -- reserved word

OPERATION: Activity during Phase IIA is controlled by the GSM chain entries. This list is prepared during Phase I and contains entries for each macro instruction, SET statement involving a global symbol, PRINT, and change of control section. (The GSM chain also contains other entries that are not pertinent to Phase IIA.)

This routine maintains an address pointer for the current control section (CCS) that indicates the dictionary item for the control section currently effective. CCS is used to establish the various values for &SYSECT as macro expansion proceeds. It is updated by this routine upon encountering a

section change entry in the GSM chain and by the control section processor (SECT) when processing a control section directive. To ensure synchronization of user-defined global variable symbols with the macros expanded during this phase, all GBL statements occurring at the user level are reprocessed. At the first redeclaration of each symbol the initial value of the item is reset to the null state.

To maintain the synchronization of global variable symbols established above, SET statements at the user level which affect global variables are also reprocessed. The value of the global symbol originally

obtained in Phase I is retained in the logical order file and is reinstated by Phase IIA.

If the assembly is in conversational mode, when all GBL, SET, and macro instruction entries in the GSM chain have been processed, Phase IIA passes over the transitive item chain in the main dictionary and extracts all symbols that remain undefined. These symbols are represented in the dictionary by transitive items that have not been completed by the insertion of the location of the matching definition. Diagnostic messages are produced for each symbol. Control then is given to the entry of LPC, which solicits continuation information from the conversational user. If the user elects to continue, control returns to the assembler at the continuation entry point, and assembly resumes with Phase IIB.

It should be noted that although only one principal subroutine apparently is called, that one -- STAN -- calls upon almost all of the routines and components described in this document for Phase I.

PARAMAC -- Macro Parameter Processor (CEVPM)

This routine creates temporary dictionaries for each outer and inner macro instruction level. (See Chart BU.)

Entry Point: CEVPM

Calling Sequence: INVOKE APARAM

Routines Called: DIAG, DLKT, DPUT

Exit: Return to STAN (via MACREF)

OPERATION: This routine is entered by STAN (via MACREF) when macro instruction statements are encountered in the GSM chain during Phase IIA.

A parameter item is created for the system variable &SYSLIST. This item contains the number of operands specified in the macro instruction (number attribute, N') and a pointer to the first of a series of trailer entries which associate each argument as a function of &SYSLIST. The &SYSLIST item is entered in the temporary dictionary before parameter items are constructed for corresponding positional macro instruction operands and the positional operands of mixed mode macro instruction. The current value of &SYSNDX is computed from a counter, and an &SYSNDX item containing the current value is placed in the temporary dictionary. An &SYSPSCT item, an &SYSSTYP item, and an &SYSECT item are entered in the temporary dictionary, and

contain respective pointers to the first PSECT item and the controlling control section item. Global pointer items are made for &SYSDATE and &SYSTIME in the temporary dictionary. The PSECT and CSECT items are located in the main dictionary. Thus, the correct unique value of system variables are maintained for varying levels of inner and outer macros.

The symbolic parameters in the macro prototype statement and the corresponding positional operands in the macro instruction are combined to form parameter items in the temporary dictionary. Each item is identified by the symbolic parameter, which is hashed and linked to an entry in the macro hash table for the current level. The type, count, and number attributes for each item are assigned as a function of the scan of the macro instruction argument. The length, type, beginning character position, and the argument character string itself are attached to the item if a corresponding entry is found in the macro instruction line. Parameter items are constructed for symbolic parameters appearing in the name field of prototype statements also. The corresponding arguments are found in the name field of the macro instructions. Sequence symbols are not considered valid arguments for this purpose, however. A null type attribute is assigned to the parameter whenever the argument is not present in the macro instruction.

For sublist operands, individual entries containing length, type, and position information are created for each operand in the sublist. The entire operand, including parentheses and separating commas, make up the argument string. Missing operands are indicated by null entries.

Parameter items with blank hash keys are created for positional operands appearing in the macro instruction for which a related prototype entry is not specified. This permits such operands to be referenced as subscripts to &SYSLIST.

Parameter items are also produced for key word operands encountered on the macro instruction line. These items are marked as "undefined". Each item is placed in "defined" status when the presence of a matching key word in the prototype statement has been verified or if a default value is specified in the prototype statement. Items are also generated for key word parameters appearing in the prototype statement for which the corresponding macro instruction operand is not present.

Error Checks:

- Invalid symbolic parameter.
- Multiple keyword operands, first one accepted.
- Positional operand follows keyword operand.
- Undefined positional prototype operand.
- Positional operand omitted.
- Multiple prototype positional operands with same name, first accepted.
- Undeclared keyword operand.
- Illegal usage reserved word.

SECTION 7: PHASE IIB

INTRODUCTION

During Phase IIB a location counter value is assigned to each symbol in the program. To arrive at a proper resolution of the location counter, the literals must also be processed and collected.

During Phases I and IIA the logical order file was flagged to indicate which source statements contain literals and which source statements defined a symbol. Associated with each statement is the length in bits of the machine-language coding that the statement represents. The lengths for each control section are accumulated and, after considering discontinuities and irregularities introduced by alignments and origins, a location counter value is assigned to each symbol as it is encountered in the logical order file.

In scanning the logical order file, literals are processed and entered into the symbol table as they occur. At LTORG statements, the literals accumulated to that point are ordered by length and are assigned location counter values. Literals not collected at the conclusion of the logical order file are collected under a LTORG generated at the end of the first control section.

Figure 20 illustrates the routine relationships to accomplish the Phase IIB functions.

Table 6 is a decision table listing the criteria for entering each Phase IIB routine.

ROUTINES

PHASE IIB -- Phase IIB Control (CEVPC)

This routine organizes the results of the initial scan over the source program so the object text can be generated in a single pass over the internal representation of the program. (See Chart BV.)

Entry Point: CEVPCX

Calling Sequence: L R15, ACEVPC
BR R15

Routines Called: CSCAN ORIGIN
DIAG POOLIT
EQUATE RESCON
LOCATE RESLIT

Exit: To PHASE IIC (CEVPD)

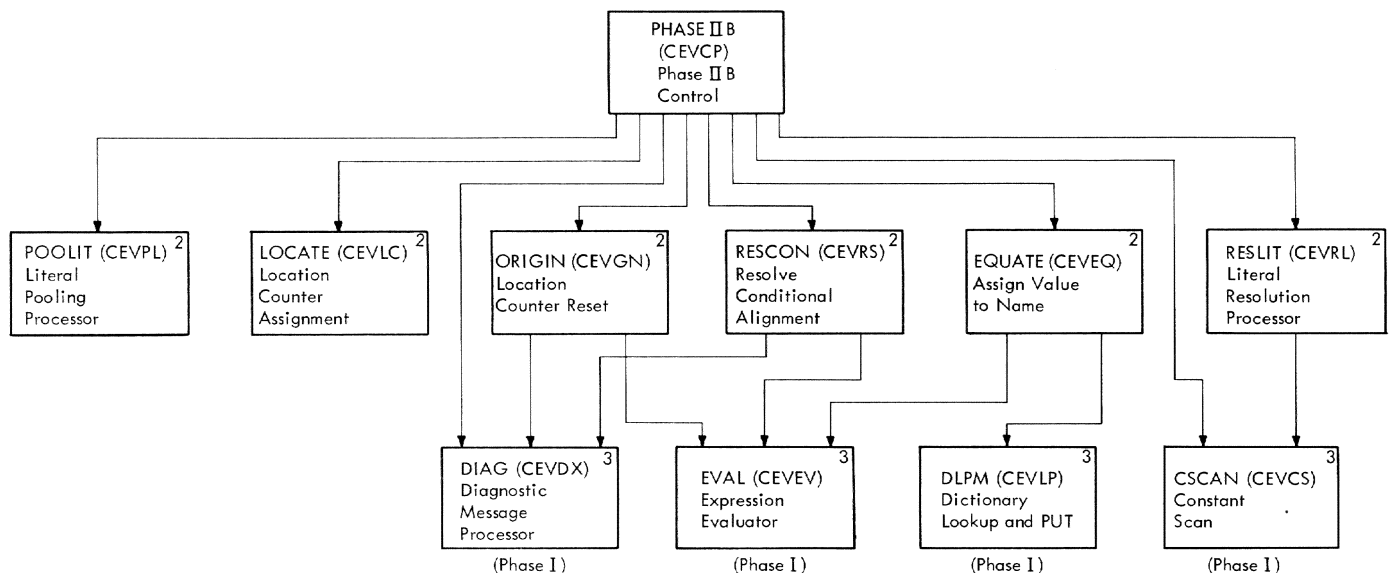


Figure 20. Phase IIB routine relationships

Table 6. Phase IIB decision table (part 1 of 2)

Routine: Phase IIB Control Level: 1			
Routine	Purpose	Called Routines	Calling Conditions
PHASE IIB (CEVPC)	Aligns all statements to the required boundary, computes page usage for each control section, resolves literal references, pools and assigns location counter values to literals, and resolves symbol definitions.	POOLIT -- CEVPL	LTORG statement encountered. A CSECT or a PSECT has been declared.
		LOCATE -- CEVLC	Statement is a named line other than an EQU.
		ORIGIN -- CEVGN	ORG statement encountered.
		CSCAN -- CEVCS	DC or DS statement not processed in Phase I.
		DIAG -- CEVDX	D115 Invalid operand field
		RESCON -- CEVRS	a. DC, DS, CNOP, or CCW statement encountered. b. Statements requiring alignment encountered.
		RESLIT -- CEVRL	Statement contains a literal reference.
		EQUATE -- CEVEQ	EQU statement which could not be evaluated in Phase I encountered.
Routine: Level: 2			
EQUATE (CEVEQ)	Constructs dictionary items for EQU statements which could not be evaluated in Phase I.	EVAL -- CEVEV	Always called.
		EATT -- CEVEV2	No location counter reference
		DLPM -- CEVLP	Operand is valid type.
		DIAG -- CEVDX	D1 Field improperly delimited D4 Duplicate symbol D6 Operand expression type invalid D48 Truncated value D69 Symbol not previously defined D115 Invalid operand field
LOCATE (CEVLC)	Assigns a location counter value to a symbol in the name field.	None	
ORIGIN (CEVGN)	Processes all ORG statements.	EVAL -- CEVEV	Always called unless ORG has been generated by a null START instruction.
		DIAG -- CEVDX	D1 Improperly delimited field D6 Invalid expression type for field D25 Attempted ORIGIN outside of control section D52 Absolute ORIGIN

Table 6. Phase IIB decision table (part 2 of 2)

Routine: Level: 2 (Cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
POOLIT (CEVPL)	Pools all literals in current literal chain. Address constants are excluded if a PSECT has been declared.	None	
RESCON (CEVRS)	Aligns DC, DS, CNOP and CCW statements and, if necessary, all other statements requiring alignment.	EVAL -- CEVEV DIAG -- CEVDX	Called for each term of a CNOP instruction. D1 Improperly delimited field D6 Invalid expression type for field D7 Invalid expression value for field
RESLIT (CEVRL)	Scans a literal, prepares a constant item for it, and enters the literal as an item in the main dictionary.	CSCAN -- CEVCS	Always called to scan literal. Called again if literal was not found in the dictionary.

OPERATION: PHASE IIB makes a single pass over the logical order file; the processing that is performed depends upon the characteristics of the entry in the logical order file. The entries may be grouped into three categories: location counter adjustments, literal operands, and normal statements. Location counter adjustments may be further subdivided into changes of control section, literal origin statements, ORG statements, and conditional storage reservation statements (such as CNOP, or DS statements with a duplication factor of zero). The processing for each of these conditions is summarized below.

Control Section Changes: The principal function of Phase IIB is to assign location counter values to symbols and literal constants. Each control section within the assembly has its own individual location counter, for which two values are maintained: the current value as it exists for any given statement, and the highest value the counter has reached during the course of processing the control section. Accordingly, at any change of control section, the current value of the location counter is saved (in the section name item in the dictionary). If this value exceeds the highest value previously saved, the highest value is also updated and saved. The current value of the location counter for the new section is retrieved and installed as the working counter for subsequent statements. The location of the section name item for the new section is also inserted in the current control section indicator.

Literal Origin Statements: If the name field of the LTOrg statement contains a symbol, the current location counter value is assigned to the symbol through the LOC-ATE routine. Then any outstanding, unpooled literals are assigned location counter values through the POOLIT routine.

ORG Statements: The ORIGIN routine is called to evaluate the operand of the ORG statement. If the expression is valid, the current location counter is set to the new value. Checks are made to preserve the highest value previously assigned if the new origin reduces the value of the counter. Conversely, a blank operand field causes the highest value to be retrieved and installed as the current location counter value.

Conditional Storage Reservations: The operation is examined to distinguish CNOP statements from DC and DS statements. The operand field of a CNOP is scanned and evaluated, and a code is set to indicate the amount of alignment required. A similar code is set upon examination of the DC or DS constant item. If necessary, an alignment entry is constructed and inserted into the logical order file to ensure that during Phase III an identical adjustment is made to the location counter. The current location counter is then adjusted to the indicated value.

Delayed Resolution of DC/DS Statements: A DC or DS entry in the LOF that does not indicate the address of a constant item represents a statement that could not be

resolved during Phases I or IIA because of lack of definitions for terms in the expressions for length, duplication, scale, or exponent. Such a statement is now processed by Phase IIB, using the constant scan routine to prepare the value item. The result is examined to see if it requires a conditional storage reservation.

Literal Operands: The presence of a literal operand in a statement causes this module to call upon RESLIT, which causes the literal to be scanned, its value determined, and an item for it to be entered in the permanent dictionary. Literals with matching texts share a basic dictionary item, but generate trailers to it if the literals are referred to from different literal origins.

Normal Statements: Normal statements are tested for the presence of a symbol in the name field, as indicated by the logical order file entry. If a symbol is present and the statement is an EQU, EQUATE is used to determine the legitimacy and the value of the operand of the EQU statement. If the statement is not an EQU, LOCATE is used to assign the current value of the location counter to the symbol. Before such assignment is made, the location counter is adjusted to a location appropriate for the alignment implied by the type of statement.

For statements that are not EQUs, the logical order file entry indicates the amount of storage required by the statement; the current value of the location counter is increased by the increment, and the next statement is obtained from the logical order file.

Page Usage Recording: A 512-byte page usage table is created for each control section other than a DSECT (or blank common), and its address carried in the control section name item. Each time the location counter for any of these sections is incremented, and the incrementing instruction is other than a DS or ORG, a bit is set in the corresponding page usage table, indicating that the page represented by bits 8-19 of the current location counter contain text. If an instruction will cause the location counter to exceed the limit of 4096 pages, that statement, plus the remaining statements (except the END statement), are made commentary. At the end of the phase, the number of bits set in each table is computed, and the total is posted in the section name item. Each section name item thus contains a total page usage count. This count is used during Phase III to compute the number of pages required to hold the output module. It is also used to compute the relative page within the text area at which the first page for the section begins.

Termination of Processing: At the end of the logical order file, the current location counter reading is preserved for whichever control section is in effect. By convention, the assembler retains the names of the first CSECT and the first PSECT statements in the program. All address constants that are literals are pooled in the PSECT. All other literals that have not otherwise been assigned are pooled in the CSECT. If no PSECT has been defined, literal address constants are also pooled in the CSECT. It is the function of the terminal processing during Phase IIB to accomplish this pooling.

A list of unpooled literals is maintained at all times. Each LTORG statement reduces the list. However, if a PSECT is present, address constants are not pooled by the LTORG routine, but are left on the list of unpooled literals. Accordingly, at the end of Phase IIB, the control routine causes the construction of logical order file entries that simulate a CSECT statement for the section which by convention is designated for this purpose (it is the first CSECT encountered). This logical order file entry is followed by another that simulates a LTORG statement. The highest value of the location counter for the designated section is reinstated, and POOLIT is called. All remaining literals that are not address constants are accordingly pooled at the end of the first CSECT. A GSM chain entry is constructed to indicate this change of section.

Phase IIB determines whether a designated PSECT exists. If it does, logical order file entries are created to simulate a PSECT and a LTORG, the location counter value is set, and POOLIT is called again, this time with an override switch set that causes the routine to accept address constants when they are encountered in the list of unpooled literals. A GSM entry is constructed to indicate this change of section also. If no PSECT exists, the override switch is set at the beginning of the phase as part of initialization. This action causes address constants to be pooled with other literals at each LTORG statement.

Page usage is recorded as described under "Page Usage Recording." Accordingly, no further processing is required at the end of the phase.

Error Checks: Invalid operand of DC or DS statement.

LOCATE -- Location Counter Assignment (CEVLC)

This routine completes the definition of relocatable value dictionary items.

Entry Point: CEVLC

Calling Sequence: INVOKE ALCATE

Routines Called: None

Exit: Normal

OPERATION: This routine is entered when a logical order file entry is encountered during Phase IIB in which bit 0 of field A is set, and the directive code does not specify an EQU statement. This indicates that a symbol exists in the name field of the source line. The logical order file entry points to the value item, and this routine inserts the current location counter value and control section identification into the item.

ORIGIN -- Location Counter Reset (CEVGN)

This routine preserves the highest value of the location counter and resets the counter to the value specified in an ORG operand. It also tests the operand for validity. (See Chart BW.)

Entry Point: CEVGN

Calling Sequence: INVOKE AORGIN

Routines Called: DIAG, EVAL

Exit: Normal

OPERATION: This routine is called by Phase IIB control when the current logical order file entry indicates the presence of an ORG statement. The logical order file entry is used to locate the operand field of the statement. EVAL is called to resolve the value of the operand. Relocatable, absolute, and null results receive further processing. Any other type of expression is rejected with a diagnostic message. Absolute values receive a diagnostic message also, but are then accepted as indicating a location counter setting relative to the current control section. Relocatable values must be simply relocatable and relative to the current control section; otherwise, they are rejected with a diagnostic message. If the absolute part of the relocatable expression is negative, its magnitude must not exceed the displacement value of the relocatable part (attempted origin below the base of the section). Null operand indicates that the origin is to be set to the highest previously attained location counter value for the current control section.

If the new origin is legitimate, and its value is less than the current reading of the location counter, the current value is compared against the highest previously

attained value. If the current value is higher, it replaces the previous high value. If the current value is not higher, it is discarded. In either case, the new origin is instated as the current value of the location counter and placed in the logical order file entry for ease of listing during Phase III.

Error Checks:

- Invalid expression type.
- Attempted origin outside of control section.
- Operand field improperly delimited.

POOLIT -- Literal Pooling Processor (CEVPL)

This routine pools all the literals accumulated by the time it is called. (See Chart CA.)

Entry Point: CEVPL

Calling Sequence: INVOKE APOLIT

Routines Called: None

Exit: Normal

OPERATION: Literal operands occurring in the source program are processed during Phase IIB by RESLIT. The literals are chained in order of occurrence to a first link that is independent of the dictionary. This routine's function is to order the literals by length, assign location counter values to each literal, and to transfer the chain, reordered by ascending location to the LTORG entry in the logical order file. If a PSECT is present in the assembly, address constants are excluded from the pool, unless an override switch is set to force their inclusion.

LTORG statements in a program are numbered in order of occurrence. Literals occurring between (or prior to the first) LTORG statements are identified as belonging to the LTORG number that is forthcoming. Accordingly, a LTORG number is maintained at all times by Phase IIB control. It is used by RESLIT to assign LTORG numbers to the individual literals, and is increased by one by this module at the termination of its processing. The reader is referred to the description of RESLIT for a further discussion of the LTORG number.

The accumulated literal chain is traversed four times, inspecting the literals on the basis of their binary text length, modulo 8, to achieve the following correspondence between length and alignment:

1st pass	0	Doubleword
2nd pass	4	Fullword
3rd pass	2, or 6	Halfword
4th pass	1, 3, 5, or 7	Byte aligned

On each pass POOLIT checks the length of the literal. If the length is appropriate for the pass, the trailer chain is traced to find a LTOrg number that matches the current one. The current location counter value is entered into the trailer and incremented by the length of the literal. The literal is removed from the independent chain and attached to the chain originated in the LTOrg item. Address constants remain on the independent chain and are not processed, unless the override switch is set. Processing continues until the independent chain is exhausted. The current location counter is adjusted to a doubleword boundary before the first pass, to ensure proper alignment.

EQUATE -- Assign Value to Name (CEVEQ)

This routine assigns a value to an EQU statement that was unobtainable during Phases I or IIA. (See Chart CB.)

Entry Point: CEVEQ

Calling Sequence: INVOKE AEQATE

Routines Called: DIAG, DLPM, EVAL, EATT

Exit: Normal

OPERATION: EVAL is called three times to process the three possible operand fields. Appropriate diagnostics are issued for null, indeterminate, literal, and error responses. The relocatable dictionary item is checked for a location counter reference; if one is present, the current LOF entry is marked for Phase III processing. Otherwise, the type attribute is placed in the dictionary. The applicable type of value item is entered into the dictionary for absolute, relocatable, and complex results. The type attribute (T') of these items is set to U, unless the symbol previously occurred on a macro reference, or the type attribute is specified in the third operand field. In the case of a macro reference, the type attribute is set to M. For absolute and complex values, the length attribute (L') is 1, unless it is specified in the second operand.

For simply relocatable values, L' is that of the equated symbol, unless the second operand is specified.

If the operand is indeterminate, the operand value is defaulted in the dictionary and a diagnostic is issued.

Error Checks:

- Field improperly delimited.
- Duplicate symbol.
- Invalid expression type.
- Truncated value.
- Symbol not previously defined.
- Invalid operand field.

RESCON -- Resolve Conditional Alignment (CEVRS)

This routine computes the amount of storage required by a statement and/or generates and inserts the appropriate alignment item in the logical order file. (See Chart CC.)

Entry Points: CEVRS, CEVRS1, CEVRS2

Calling Sequence: INVOKE ARSxxx
xxx may be:
CON
100
870
... error return
... normal return

Routines Called: DIAG, EVAL

Exit: Normal
Error - Invalid operand of a CNOP, or location counter overflow.

OPERATION: This routine is called by Phase IIB when it has encountered a data reservation or constant statement whose precise effect upon the location counter could not be determined by STAN. This case is signaled by bit 2 of field A of the standard logical order file entry and results from a CNOP or CCW command or a DC or DS statement. An entry is also made when the location counter is not aligned for any instruction requiring byte or halfword alignment.

For CNOP commands, the location counter is aligned to a halfword. Then the operand of the CNOP is evaluated, and the amount of additional alignment is placed in the length field of the LOF entry.

For CCW commands, the location counter is aligned to a doubleword. An alignment entry is constructed for the difference, if any.

For DC/DS statements, the constant type is analyzed. Type H, S, and Y cause alignment to a halfword; types F, E, A, V, and Q

cause alignment to a fullword; and type D causes alignment to a doubleword. Other types cause alignment to a byte. If no alignment is required (multiple operands for a bit-length specification), a return is made to Phase IIB control.

Error Checks: Invalid CNOP operands.

RESLIT -- Literal Resolution Processor (CEVRL)

This routine scans a literal as if it were a normal DC statement operand, prepares a constant value item for it, and enters the literal as an item in the main dictionary. (See Chart CD.)

Entry Point: CEVRL

Calling Sequence: INVOKE ARSLIT

Routines Called: CSCAN

Exit: Normal

OPERATION: This routine obtains the precise location of the literal operand from the logical order file entry. It then calls upon CSCAN to scan the operand to prepare a constant value item for it. CSCAN is cognizant of a literal operand mode and ensures that restrictions applicable to literals are observed; these include prohibition of multiple operands, and zero duplication factors.

This routine then extracts those characters of the text to which the hashing algorithm is to be applied. The first eight characters (excluding the =) are used if the literal is that long. If it is shorter, the entire text (excluding the right delimiter) is used.

If the hashed key fails to match a literal item already in the dictionary, a new item is constructed and entered. A trailer for this item is added which identifies the LTORG number under which the literal was referenced, and the literal is added to the list of unpooled literals.

If the hashed key matches the key of a literal item already in the dictionary, the

entire text is compared. Literals are pooled on the basis of matching text. If the text is not identical, processing proceeds as described in the preceding paragraph. If the text matches, the literal itself is tested to see whether it is an address constant containing a reference to the location counter (notation of asterisk). If so, a trailer item is added that contains the current value of the location counter and the current LTORG number. If the literal is not such an address constant, the trailer chain is checked to see whether a trailer is present for the current LTORG number. If not, such a trailer is constructed and added to the literal item.

Comments: Literals are pooled on the basis of identical source text, but are also collected at each LTORG statement. Thus, the same literal may appear in multiple literal pools, depending upon its occurrence in relation to multiple LTORG statements. To control this duplication, Phase IIB maintains a LTORG number, which is the number of the next anticipated LTORG statement. The LTORG number is set to one by Phase IIB initialization and is increased by one each time the LTORG processor is called. Literals processed by RESLIT are assigned the current LTORG number. Thus, if a program contains no LTORG statement, all literals are assigned LTORG number one; Phase IIB control always generates a LTORG at the termination of processing to collect all unpooled literals, and this pool will receive all literals by default.

When LTORG statements are present, however, the value of the LTORG number changes during the course of Phase IIB.

If this routine finds a literal that matches one previously encountered, it checks further to see if the dictionary item contains a trailer item with the current LTORG number. If not, the literal was previously pooled under another LTORG and must be duplicated. The trailers to the literal item in the dictionary represent the number of duplications required to place the literal in multiple pools.

INTRODUCTION

The requirements of TSS/360 are such that the machine-language output of the assembler must be generated in order by control section during Phase III. However, for the convenience of the assembler language programmer, control sections may be written discontinuously. In particular, USING and DROP statements have an effective range which is not related to the range of a control section. Phase IIC is responsible for preparing tables that summarize the USING status of all registers at each discontinuity in a control section, so that

proper values will be available when the control section is processed in continuous order in Phase III.

In addition, ENTRY statements are processed by associating the name of each entry point with the proper control section for its definition and R-type addressability.

Figure 21 illustrates the routine relationship to accomplish the Phase IIC functions. Table 7 is a decision table listing the criteria for entering each Phase IIC routine.

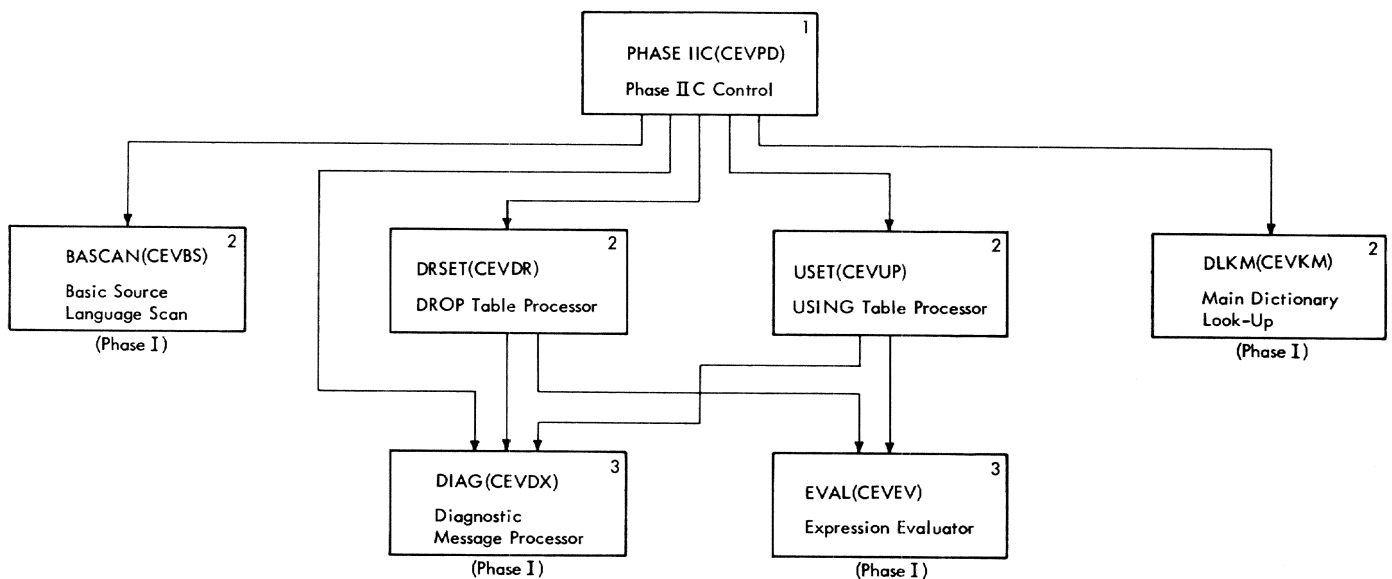


Figure 21. Phase IIC routine relationships

Table 7. Phase IIC decision table (part 1 of 2)

Routine: Phase IIC Control Level: 1			
Routine	Purpose	Called Routines	Calling Conditions
PHASE IIC(CEVPD)	Tabulates the status of PRINT control, LTORG numbers, and USING registers in relation to each control section when a section was originally written discontinuously.	USET -- CEVUP	USING GSM entry encountered in GSM chain.
		DRSET -- CEVDR	DROP GSM entry encountered in GSM chain.
		BASCAN -- CEVBS	ENTRY GSM entry encountered in GSM chain.
		DLKM -- CEVKM	ENTRY GSM entry encountered in GSM chain.

Table 7. Phase IIC decision table (part 2 of 2)

Routine: Level: 1 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
		DIAG -- CEVDX	D1 Improperly delimited field D18 Entry operand defined in invalid section type. Considered not an entry. D56 Entry point name duplicates module name D115 Invalid operand field
Routine: USING Table Processor Level: 2			
USET (CEVUP)	Updates the status of the using-register table based on information from a USING statement. To accommodate GR0 being used as a base register, the Using-Register Tables were lengthened by two words (same format as the two-word entries for other registers). The first bit of word 1 in the U-R table is no longer used to indicate GR0 availability as a base register.	EVAL -- CEVEV DIAG -- CEVDX	Always called. D1 Improperly delimited field D6 Invalid expression type for field D7 Invalid expression value for field D12 Duplicate use of register D13 Invalid expression for base register usage D14 Required operand missing D115 Invalid operand field
DRSET (CEVDR)	Updates the status of the using-register table based on information from a DROP statement.	EVAL -- CEVEV DIAG -- CEVDX	Always called. D1 Improperly delimited field D6 Invalid expression type for field D7 Invalid expression value for field D12 Duplicate use of register

ROUTINES

PHASE IIC -- Phase IIC Control (CEVPD)

This routine tabulates the status of PRINT control, LTORG numbers, and using-registers in relation to each control section. It also associates the operands of ENTRY statements with the names of control sections so R-type addressability is established. (See Chart CE.)

Entry Point: CEVPDX

Calling Sequence: L R15, ACEVPD
BR R15

Routines Called: BASCAN DLKM EVAL
DIAG DRSET USET

Exit: To PHASE III (CEVPE)

OPERATION: Construction of the output module requires Phase III to process each control section contiguously. This routine is required to maintain compatibility with OS/360 definitions of PRINT, LTORG, USING, and DROP statements, while processing in control section order.

PRINT, USING, and DROP statements are entered into the GSM chain during Phases I and IIA. GSM entries for LTOrgs are added to the chain in Phases I, IIA, and IIB. By Phase IIC, only section headings, PRINT, USING, LTORG, ENTRY, and DROP statements remain in the GSM chain. This routine constructs and maintains a full-fledged using table as it searches the GSM chain.

If an internal symbol dictionary (ISD) is to be produced following Phase III, an ISD list of using table locations is established.

Listing specifications and the current LTORG number are maintained in the first word of the table and are updated by each PRINT or LTORG statement. When a GSM link for a section heading is encountered, a using table item is inserted in the GSM chain to point to the table as it now stands. This table reflects the current status of all using registers at the time of a section break.

After each USING, DROP, or section change, the table is copied into a new area, and control pointers are updated to address the new table. Processing continues as before until the end of the GSM chain. The PRINT, LTORG, and ENTRY links are removed from the GSM chain as processed. The entries for USINGs and DROPS are replaced by pointers to the copied using table.

Phase III uses the GSM chain as a guide to processing order. In following the continuations of a control section, the GSM link for the section heading is immediately followed by a link pointing to the current using table, which reflects the status of using registers, PRINT control, and the current LTORG number at the point of continuation. The using tables are constructed in working segment 1.

During Phases I and IIA the presence of ENTRY statements is also marked on the GSM chain. This module processes this chain concurrently for both ENTRY and USING/DROP statements. At this time definitions are available for any symbol which may legitimately appear as an ENTRY operand.

This section name within which the ENTRY occurs is also known, since the GSM chain includes section names which this module records in the current control section address (CCS).

If the ENTRY occurs within a named section which is not a DSECT or unnamed CSECT, entry-operand items are constructed in the main dictionary and chained to the item for the named section currently in control.

This produces a definition which is capable of R-type references. ENTRY statements may not appear in DSECTs or unnamed CSECTs.

Error Checks:

- Improperly delimited field.
- Entry operand defined in invalid section type. Considered not an entry.
- Entry point name duplicates module name.
- Invalid operand field.

USET -- USING Table Processor (CEVUP)

This routine updates the status of the using-register table currently in effect. (See Chart CF.)

Entry Point: CEVUP

Calling Sequence: INVOKE AUSET

Routines Called: DIAG, EVAL

Exit: Normal

OPERATION: This routine is called during Phase IIC when the current GSM entry indicates the presence of a USING statement. It first resets a series of indicators used to check for duplicate register specifications. EVAL is called to evaluate the first operand, which is the base value for the using-registers. Absolute and relocatable expressions are acceptable. The relocatable expression may consist of a single external or internal symbol, plus or minus any absolute value. Indicators are set to indicate whether the table entry is to be in absolute, relocatable, or external format. The base value is set accordingly.

Each of the remaining operands is submitted in turn to EVAL. The expression must be absolute, less than 16, and must not duplicate another operand. If legal, the table entry for the specified register is constructed. The value 4096 is then added to the base value for each legal operand after the first until the list of operands is completed.

The logical order file entry for the USING statement is completed. Indicators are inserted showing the type of base expression and its value.

Error Checks:

- Duplicate register usage.
- Invalid expression for base register value.
- Invalid expression for register specification.
- Improperly delimited operands.
- Invalid expression value for register specification.
- Required operand missing.
- Invalid operand.

DRSET -- DROP Table Processor (CEVDR)

This routine updates the status of the using-register table currently in effect. (See Chart CG.)

Entry Point: CEVDR

Calling Sequence: INVOKE ADRSET

Routines Called: DIAG, EVAL

Exit: Normal

OPERATION: This routine is called during Phase IIC when the current GSM entry indicates the presence of a DROP statement. A check is made to see if the increment to the operand field is zero; if so, all registers are set to be dropped. A series of indicators used to check for duplicate register specifications is first reset.

EVAL is called to evaluate the expression for each of the operands. Each expression must be absolute, less than 16, and must not duplicate another operand. If legal, the table entry for the specified register is marked as not used.

Error Checks:

- Duplicate register usage.
- Invalid register expression.
- Improperly delimited operands.
- Invalid register value.

INTRODUCTION

During Phase III, the following is produced:

- Machine-language text.
- Control section dictionaries and list of external symbols (used by LPC to establish alias names when the program module is stowed).
- Optionally a list data set, containing listings of:

Source statements
Object program

The machine-language text, control section dictionaries, and list of external symbols are left in virtual storage at the conclusion of the assembly. The language processor control (LPC), using the list of external symbols prepared by the assembler, performs the input/output functions necessary to incorporate the text and dictionaries into a partitioned data set.

Source statements and object program listings are optional items; the user may request either one listing or both. Listings requested by a conversational user will be filed automatically in a VISAM list data set unless the user requests a printout at his terminal. Listings requested by a nonconversational user will be printed on SYSOUT through a GTWRC macro unless the user requests a list data set instead.

The encoded information in the logical order file, and the operands of the original and generated source language statements are used to construct machine-language text and control section dictionaries. The object listing is produced as a by-product of this activity. The listing also contains diagnostic messages for uncorrected errors.

The machine-language output is distributed between text and control section dictionaries. The text contains machine-language instructions; the virtual storage pages containing the text ultimately maps onto pages in the target virtual storage of the assembled program.

The control section dictionaries contain information about external symbol definitions, external symbol references, and the relocation properties of the text, all organized by control section.

The listing data set contains a line image listing suitable for printing on an external device by the BULKIO facilities of the command language.

Figure 22 illustrates the routine relationships to accomplish the Phase III functions. Table 8 is a decision table listing the criteria for entering each Phase III routine.

ROUTINESPHASE III -- Phase III Control (CEVPE)

This routine controls the final processing of all instructions. It organizes the program by control section, produces the necessary binary output text and relocation information for the object program, and provides listings of the source and object programs. (See Chart CH.)

Entry Point: CEVPEX

Calling Sequence: L R15,ACEVPE
BR R15

Routines Called: CCWTXT EVAL
CSDPR LIST
DCTXT LITXT
DIAG MOPR
ENDPR SLLS

Exit: To PHASE IV (CEVPF)

OPERATION: Upon entry, a source program listing is prepared if one has been specified.

The page usage estimated for the output is then calculated, and VMGET is called to procure output working storage and storage needed for the program module dictionary and external name list. The module heading is constructed and completed as far as possible.

The GSM chain is used to put the program into order by control section. Within each section, the logical order file controls the order of processing. Each statement represented in the logical order file is processed by an appropriate open or closed subroutine. As each control section is completed, the control section dictionary for that section is added to the PMD. At the end of the last control section, diagnostic messages for the entire assembly are printed.

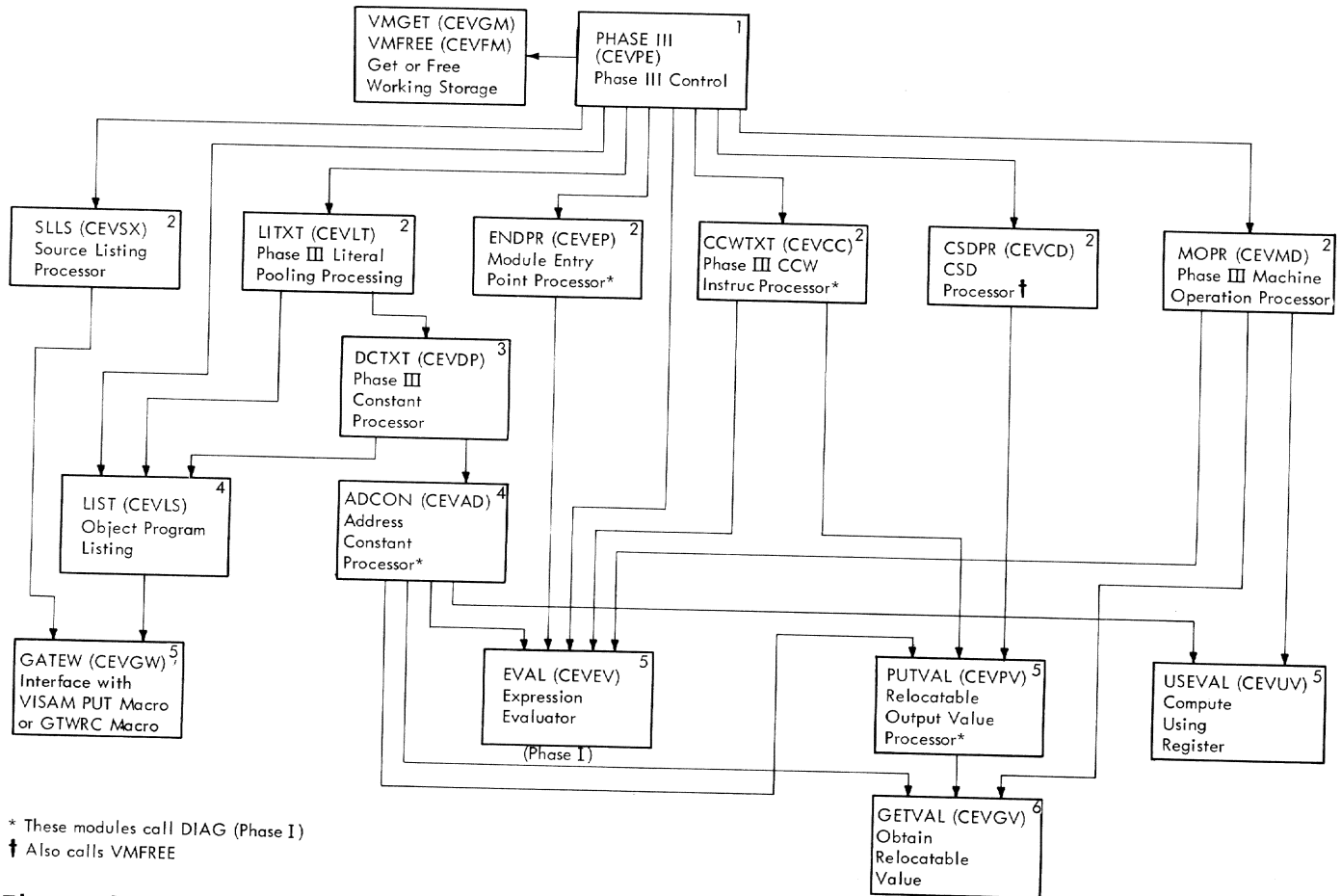


Figure 22. Phase III routine relationships

Table 8. Phase III decision table (part 1 of 4)

Routine: Phase III Control Level: 1			
Routine	Purpose	Called Routines	Calling Conditions
PHASE III (CEVPE)	To control the final processing of all instructions. It organizes the program by control section, produces text and relocatable information and provides listing.	LIST -- CEVLS	Always called to output a line. Not called for a DC, LTORG, or alignment entry with no text required.
		SLLS -- CEVSX	Source language listing option is on.
		ENDPR -- CEVEP	Always called.
		CCWTXT -- CEVCC	CCW instruction encountered.
		MOPR -- CEVMD	Machine instruction encountered.
		LITXT -- CEVLT	LTORG instruction encountered.
		DCTXT -- CEVDP	DC or DS instruction encountered.

Table 8. Phase III decision table (part 2 of 4)

Routine: Level: 1 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
		CSDPR -- CEVCD	Called if program contains control sections other than DSECT.
		EVAL -- CEVEV (Phase I)	SPACE, EQU, ORG or USING instruction encountered.
		DIAG -- CEVDX (Phase I)	D115 Invalid operand of SPACE instruction.
		VMGET	Gets pages for text, PMD, and external names list.
		VMFREE	Free storage obtained in Phase III control if insufficient storage available for PMD and external names list.
Routine: Level: 2			
CCWTXT (CEVCC)	To evaluate the operand field of a CCW instruction and to create binary output in the text module.	EVAL -- CEVEV (Phase I)	Always called.
		DIAG -- CEVDX (Phase I)	D6, D7 Invalid expression D1 Invalid delimiter D10 Attempted read into literal D20 Invalid flag field
		PUTVAL -- CEVPV	Data address is complex or relocatable.
CSDPR (CEVCD)	Retrieves relocation modifiers and reference items in temporary storage and produces a final output CSD.	PUTVAL -- CEVPV	Complex relocatable item encountered.
		VMFREE -- CEVFM	Free unused external names list pages.
ENDPR (CEVEP)	To analyze the operand of the END statement and to complete the output PMD heading relative to the module entry point.	EVAL -- CEVEV (Phase I)	Always called.
		DIAG -- CEVDX (Phase I)	D11 Invalid operand on END statement D6 Invalid expression type
LITXT (CEVLT)	To place the binary text for the literals in the given pool into the output module.	DCTXT -- CEVDP	Literals found in a given pool.
		LIST -- CEVLS	Always called.
MOPR (CEVMO)	To evaluate the operand field of a machine instruction and to create binary output in the text module.	EVAL -- CEVEV (Phase I)	Always called.
		USEVAL -- CEVUV	Always called unless RR-type instruction.
		GETVAL -- CEVGV	Operand is relocatable or literal.

Table 8. Phase III decision table (part 3 of 4)

Routine: Level: 2 (cont'd)			
Routine	Purpose	called Routines	Calling Conditions
		DIAG -- CEVDX (Phase I)	D1 Invalid delimiter D14 Required operand missing D39, Length of multiplier/ D41 divisor too large D40 Length field exceeds 3 (SLT instruction) D42 Specified register dup- licates implicit register D33, Floating point register D34 required D35 Even numbered register required D36, Operands not on proper D37, boundary D38 D6, Invalid expression D7 D10 Attempted store into literal D30 Implied length too large D120 Attempted explicit register specification
SLLS (CEVSX)	Prepare source listing.	GATEW -- CEVGW	Always called.
Routine: Level: 3			
DCTXT (CEVDP)	Places the binary text for the constant into the output module.	ADCON -- CEVAD EVAL -- CEVEV LIST -- CEVLS	Address constant encountered in LOF chain. A DS statement containing an expression and a cross-refer- ence listing is desired. 1. First duplication of a DC or DS. 2. nth duplication of a DC or DS (n>1), and DATA option previously specified on a PRINT statement. 3. Bit length constants.
Routine: Level: 4			
ADCON (CEVAD)	Process address constants.	DIAG -- CEVDX (Phase I)	D1 Improperly delimited field D6 Invalid expression type for field D7 Invalid expression value for field D32 Absolute value of dis- placement too large D44 Address constant contains literal D48 Truncated value D61 Value of length modifier invalid for type of constant D132 Improper operand for Q-type adcon

Table 8. Phase III decision table (part 4 of 4)

Routine: Level: 4 (cont'd)			
Routine	Purpose	Called Routines	Calling Conditions
		PUTVAL -- CEVPV	Relocatable A-type adcon encountered.
		USEVAL -- CEVUV	S-type adcon encountered.
		EVAL -- CEVEV (Phase I)	Always called.
		GETVAL -- CEVGV	Relocatable A-type adcon encountered.
LIST (CEVLS)	Prepare print lines for object program listing.	GATEW -- CEVGV	Called if object listing has been requested.
Routine: Level: 5			
GATEW (CEVGV)	Interfaces the assembler with listing output macros.	VISAM PUT or GTWRC macro instructions.	Always called.
USEVAL (CEVUV)	Reduces a relocatable location or absolute value to a base register and displacement value.	DIAG -- CEVDX (Phase I)	D29 No base register available for implicit addressing D32 Absolute value of displacement too large
PUTVAL (CEVPV)	Prepare relocation information and binary text for address constants and the data field of CCW instructions; this information ultimately inserted in a CSD in the PMD.	DIAG --CEVDX (Phase I) GETVAL -- CEVGV	D48 Truncated value D111 Address constant refers to DSECT Always called.
Routine: Level: 6			
GETVAL (CEVGV)	Get value from dictionary.	None	

The module heading is completed, and the control section dictionaries for each section are scanned to prepare a list of external names so that LPC can stow them when disposing of the output.

Error Checks:

- Invalid operand of a SPACE instruction.
- Other error checks performed by the component subroutines.

Comments: The language permits the generation of empty pages of virtual storage. Empty pages are not represented in the text. Accordingly, when not in the list only mode (used for macro definitions and DSECTS), prior to the processing of any statement that may generate text, the virtual location counter for the object pro-

gram is correlated to the position in the text storage area that the generated text will occupy. A table, representing the virtual storage of the object program control section, is maintained for this purpose. It is initially set to unused (all bits on in all words). The first time a generative instruction refers to a page of virtual storage, a corresponding page of text is allocated for it. The relative page number of the text is inserted in the virtual storage map. Subsequent location counter references to the same page of virtual storage cause the binary output to be placed on the same page of text. At the end of the control section, the virtual storage table indicates how many pages of text have been generated and which virtual storage pages they represent. That portion of the table containing entries is then placed in the PMD as a guide for the loader.

SLLS -- Source Listing Processor (CEVSX)

This routine prepares a listing of the input source lines in their original order and format, together with the sequence number assigned to the statement by the line data set facilities. (See Chart CI.)

Entry Point: CEVSX

Calling Sequence: INVOKE ASLLS

Routines Called: GATEW

Exit: Normal

OPERATION: This routine is called during Phase III if the source listing option has been taken. Input source lines and their associated sequence numbers are linked together in virtual storage during Phase I of the assembler. The chain of lines are followed, the sequence number is edited for printing, and the original source line is printed without further editing. Processing is terminated by the occurrence of the last link in the source line chain. The VISAM PUT macro is used to place the edited line in the listing data set or the GTWRC macro is used to place it on SYSOUT.

GATEW -- Interface with VISAM PUT or GTWRC Macro (CEVGW)

This routine provides the interface to store list lines in a listing data set with the VISAM PUT macro, or put them immediately on SYSOUT with the GTWRC macro. (See Chart CJ.)

Entry Point: CEVGW

Calling Sequence: INVOKE AGATEW

Input Parameters:

R1 Location of line to be placed in the listing data set. A length of 133 characters is assumed.

Routines Called:

- Internal - None
- External - PUT macro (VISAM) or GTWRC macro

Exit: Normal

OPERATION: This routine is supplied the location of the line to be placed in the listing data set. It constructs a sequential key to satisfy VISAM requirements, appends the key to the item, and then tests to determine whether the line is going to a VISAM list data set. If so, a PUT macro places the logical record into the data

set. If the user requests listings on SYSOUT, a GTWRC macro generates code to place his line on SYSOUT. A return is then made to the caller.

ENDPR -- Module Entry Point Processor (CEVEP)

This routine analyzes the operand of the END statement and completes the output PMD heading relative to the module entry point. (See Chart CK.)

Entry Point: CEVEP

Calling Sequence: INVOKE AENDPR

Routines Called: DIAG, EVAL

Exit: Normal

OPERATION: This routine calls EVAL to evaluate the operand of the END statement. If the expression is null, the module entry point is set to zeros. If the expression is relocatable, a reference entry and a modifier are added to the module heading. The length of the module heading is completed, and the location of the first control section dictionary is established.

Error Check: Invalid expression type.

MOPR -- Phase III Machine Operation Processor (CEVMO)

This routine causes the operand field of the statement to be evaluated and creates corresponding binary output in the text portion of the output program module. (See Chart CL.)

Entry Point: CEVMO

Calling Sequence: INVOKE AMOPR

Routines Called: DIAG GETVAL
EVAL USEVAL

Exit: Normal

Error - Same exit as normal; corresponding text for unprocessed fields set to 0.

OPERATION: This routine is called during Phase III when the current logical order file entry indicates that a machine instruction statement is to be assembled. The address in the output text that the instruction is to occupy is calculated prior to entry. The instruction length is determined from the logical order file, and the bytes to be occupied by the assembled text are set to zero. The operation code is transferred from the logical order file entry to the text.

Processing proceeds according to the instruction type: RR, RR with extended M1 field, RR with only one register, RR with immediate value, RX, RX with extended value, RS with explicit R3 field, RS without R3 field, SI with immediate value, SI without immediate value, SS with two length fields, and SS with one length field.

The syntax of the operand field for each instruction type is evaluated and checked for validity. As each component field of the instruction is evaluated, the corresponding binary output is placed in the text. When the text has been completed all instructions are checked against the machine operations requirement table to diagnose alignment errors and improper register usage.

Relocatable operands are submitted to USEVAL, which attempts to reduce the relocatable symbol to a base register displacement value. Successful reduction causes the resulting values to be placed in the specified B and D fields of the instruction text. Unsuccessful reduction leaves the B and D fields of the text zero and produces a diagnostic message.

The location counter value of relocatable operands, including literals, is obtained by GETVAL. GETVAL retrieves the section number and displacement for ordinary symbols; it also searches a literal item for the proper trailer in order to retrieve a location counter value appropriate for the current reference.

Error Checks:

- Improperly delimited field.
- Improper type of expression for field.
- Value too large for field.
- Attempted store into literal.
- Length field exceeds 3 (SLT instruction).
- Improper register specification.
- Improper storage alignment for instruction.
- Required operand missing.

GETVAL -- Obtain Relocatable Value (CEVGV)

This routine helps in the preparation of relocation information to obtain the section number and location counter displacement of a relocatable symbol. It also indicates whether the symbol was external,

complex, or simply relocatable. (See Chart CM.)

Entry Point: CEVGV

Calling Sequence: INVOKE AGTVAL

Input Parameters:

- R1 Location of dictionary item, expressed as a 24-bit increment to the base of working segment 2.
- R5 Pointer to a one byte field containing a 'Q' if a Q-con is being processed, or non-'Q' if any other type of constant.

Routines Called: None

Exit: Normal

Output Parameters:

- R0 Mode indicator:
 - 0 = Relocatable
 - 1 = Complex
 - 2 = External
- R1 If relocatable, 8-bit section number and 24-bit displacement; if external, reference number in bits 0-15 and using table number in bits 16-31; if complex, the absolute portion of the complex value, expressed as a 32-bit signed number.
- R2 If complex, the length in words of the list of relocation words in the complex definition.
- R3 If complex, the location (32 bits) of the first word in the relocation list.

OPERATION: This routine is called during the processing of machine instructions. It is supplied the location of a dictionary item as an input parameter. Normally, the location of this item is obtained from the RLD list produced by EVAL in response to a relocatable expression.

If, upon examining the dictionary item, the symbol is external, the external exit is taken.

If the item is relocatable, the section number and displacement of the symbol is supplied and the relocatable exit is taken.

If the item is a literal, the literal trailer chain is searched until a trailer is found that matches the LTRG number currently in effect. When a matching trailer is found, the trailer is tested to see if

the literal is an address constant referring to the location counter. If so, the location counter in the trailer is tested to see if it matches the current value of the object program location counter. If not, the search continues until the matching trailer is found. If the literal is not this special kind of address constant, the first matching trailer found is the correct one. In either case, the section number and displacement contained in the correct trailer represent the appropriate relocatable value for the definition of the literal. This information is supplied, and the relocatable exit is taken.

If the item is a complex definition, the length and location of the RLD modifiers in the complex definition and the absolute value portion of the definition are supplied, and the complex exit is taken. If the item is a section name and a Q-con is being processed, the external mode and exit are taken.

USEVAL -- Compute Using Register (CEVUV)

This routine reduces a relocatable location or absolute value to a base register and a displacement value. (See Chart CN.)

Entry Point: CEVUV

Calling Sequence:

```

INVOKE AUSVAL
... error return (four bytes)
... normal return

```

Input Parameters:

- R0 Type code: 0 for relocatable; 1 for absolute; 2 for external.
- R1 32-bit value if absolute; 8-bit section number and 24-bit displacement if relocatable; reference number in bits 0-15 and using table number in bits 16-31, if external.
- R2 32-bit signed absolute portion of relocatable or external expression.

Routines called: DIAG

Exits: Normal

```

R0 Bits 0-15, zero
   Bits 16-19, register number
   Bits 20-31, displacement

```

```

Error
R0 All zero

```

OPERATION: This routine is called by MOPR during Phase III. It accepts a value and a type indicator as input. The type indica-

tor specifies absolute, external, or relocatable base type; the value is the absolute value: zero, if external; or a section number and displacement, if relocatable.

Depending upon the type, a suitable search key is constructed. The current using register table is searched, linearly, for an entry of the correct type which will yield a displacement of 4095 or less. If such an entry is found, a record of it is made. The expanded size of the Using-Register Table (normal treatment of general register 0) is taken into consideration. Bit 1 of word 1 of the U-R Table is no longer examined. The search continues, however, since the established algorithm is to use the base register that produces the lowest displacement and, in the case of two or more registers that produce identical displacements, to use the highest numbered register.

At the completion of the search, diagnostic messages are produced if required; or, if the search was successful, the appropriate register number and displacement value are presented as output.

Error Checks:

- Absolute value of displacement within range.
- Base register available for implicit addressing.

LIST -- Object Program Listing (CEVLS)

This routine provides a line on the object program listing appropriate to the statement type represented by the current logical order file entry. (See Chart CO.)

Entry Point: CEVLS

Calling Sequence: INVOKE ALIST

Routines Called: GATEW

Exit: Normal

OPERATION: The current logical order file entry and the location of any binary text generated for the statement represented by the logical order file entry are used to prepare a suitably formatted line for the object program listing.

If PRINT control is set to OFF, no processing is performed. If it is set to ON, information relative to the binary text on the left side of the listing and information relative to the source statement on the right side is printed. Each half of the line is prepared independently in order

to accomodate continued source lines, extended binary constants, and the like.

A separate processing path is used for each of the principal assembler operations to insure appropriateness of format.

The PUT macro in VISAM is used to dispose of each line if it is going to a list data set. The GTWRC macro is used if it is going to SYSOUT. The line is 132 characters and is preceded by an ASA FORTRAN standard print control character: blank for single space, 0 for double space, and 1 for page eject.

CCWXTX -- Phase III CCW Instruction Processor (CEVCC)

This routine evaluates the operand field of the CCW statement and creates corresponding binary output in the text portion of the output program module. (See Chart CP.)

Entry Point: CEVCC

Calling Sequence: INVOKE ACCWTX

Routines Called: EVAL, PUTVAL, DIAG

OPERATION: This routine is called during Phase III when the current logical order file entry indicates that a CCW assembler instruction statement is to be assembled. The address in the output text which the CCW is to occupy is calculated prior to entry. Adjustment will have been made to a doubleword boundary. The eight bytes of text are then set to zero.

The syntax of the four operands is evaluated and checked for validity. As each component field of the instruction is evaluated, the corresponding binary output is placed in the text.

PUTVAL is called for relocatable data address operands, including literals where valid, to create the necessary relocation dictionary information required to modify the text of a relocatable field.

Error Checks:

- Illegal command code.
- Improperly delimited operands.
- Improper values for data address, flag, and count fields.
- Attempted read or read backward into literal.

PUTVAL -- Relocatable Output Value Processor (CEVPV)

This routine prepares the relocation information that will ultimately be placed in the control section dictionary of the program module dictionary and the text to which the relocation applies. (See Chart CQ.)

Entry Point: CEVPV

Calling Sequence: INVOKE APTVAL

Input Parameters:

- R1 Absolute value portion of expression (32-bits signed).
- R2 Length of relocation input list in bytes.
- R3 Location (32 bits) of relocation input list.

Routines Called: DIAG, GETVAL

Exit: Normal

OPERATION: This routine is called during the processing of address constants and the data address field of CCW instructions. It is supplied with the length and location of the desired text, the address and length of an RLD list (normally produced by EVAL) and the absolute value portion of the expression. The format of RLD items is shown in Appendix B of Dynamic Loader PLM.

The RLD list produced by EVAL is the same as that contained in a complex value dictionary item.

This routine transforms each entry in the RLD into a temporary relocation item. The temporary relocation items are converted into true relocation modifiers by CSDPR at each section change encountered by Phase III control.

A working value for the text is first established. This value is initialized with the absolute value portion of the relocatable expression. It is modified by the displacements of subsequent relocatable symbols and ultimately is moved (and truncated if necessary) to fit into the text output.

The items in the RLD list that was supplied as input are processed one by one. GETVAL is called to provide section numbers and displacements for relocatable symbols. The displacement is added to or subtracted from the text value according to the operator code. The section number is used to find the reference number of the section

name. If a reference number has not been assigned to the section name during the processing of the current control section, one is assigned, and a temporary reference item is constructed in working storage. The reference items are also processed by CSDPR when the section is complete.

The temporary relocation item is constructed, using the length and location of the text, the operator code, and the reference number of the relocatable value. For a complex definition, one temporary relocation item is prepared for each relocatable quantity which is required to express the definition.

GETVAL may indicate that a given symbol is external. In that case, the external name item is located in the dictionary (its location is given in the input relocation list) and the reference number of the external symbol is obtained. If no number has been assigned, one is assigned, and a temporary reference item is created. Processing proceeds as for a relocatable item.

GETVAL may also indicate that a given symbol has a complex definition. If so, an entry for a pushdown list that indicates the length and position of the current relocation input is constructed. The absolute value supplied by GETVAL is combined with the working text value. The length and location of the relocation list supplied by GETVAL is instated as the current list, and processing continues. Complex definitions may contain complex definitions. At the end of a relocation list, this module pops up its pushdown list and resumes processing the outer definition. If the pushdown list is empty, all terms of the relocatable expression have been processed.

The working value of the text is moved to the location desired, truncating it, if necessary, to fit the field length. If significant bits are lost by this truncation, a diagnostic message is produced.

Error Checks:

- Truncated value.
- Address constant refers to DSECT.
- Illegal use of DXD symbol.

DCTXT -- Phase III Constant Processor (CEVDP)

This routine places the binary text for the constant of a DC statement into the output module and calls LIST to output an object program listing line for both DC and DS statements. The text and relocation

values for address constants not previously obtained are resolved during this processing. Cross-references are processed for DC statements, and for DS statements if a cross-reference listing has been requested. (See Chart CR.)

Entry Point: CEVDP

Calling Sequence: INVOKE ADCTXT

Routines Called: ADCON, LIST, EVAL

Exit: Normal

OPERATION: This routine is called by the Phase III control module (CEVPE) when the current LOF entry indicates that a DC or DS statement is to be processed. If expressions have been found in the modifiers of a DS statement and a cross-reference listing has been requested, EVAL will be called to evaluate the expressions. LIST will then be called to output an object program listing line for the DS statement before DCTXT returns to the caller.

For a DC statement, the constant item associated with the current LOF entry is examined, and a duplication count for the text is established. If the constant is an address constant, ADCON is called to produce text and relocation information for the constant. If the constant is not an address constant, its value is retrieved from the constant item and moved to the text location. For bit-length constants the text location is bit oriented, special shift and move techniques are used to pack the data into the bit-aligned field. Movement of data into the text is repeated until the duplication factor is reduced to zero. Printed output is generated on each duplication when the DATA print option is specified, except for bit-length fields.

For bit-length fields, the next LOF entry is tested when all duplications of the current constant are complete. If the next LOF entry indicates a multiple-operand bit-length constant, the bit-oriented text location is maintained at its current updated value so that the next constant may be packed at the next adjacent bit. If the next LOF entry does not indicate a multiple-operand bit-length constant, any alignment entry that may be present in the LOF is used to fill out the current field of the next byte boundary with zero bits. The entire bit-length constant is then printed.

The multiple operands for non-bit-length constants are processed by successive entries to this module. The LIST module suppresses the source line on all continuations.

ADCON -- Address Constant Processor (CEVAD)

This routine prepares the binary output text of a constant and any information pertaining to the relocation properties of the value. (See Chart CS.)

Entry Point: CEVAD

Calling Sequences:

INVOKE AADCON
... error return (4 bytes)
... normal return

Routines Called: DIAG PUTVAL
EVAL USEVAL
GETVAL

Exit: Normal

Error - various errors occurring in operand of address constant.

OPERATION: This routine is called during the processing of literals and constants in Phase III. The initial location of the output text is established, and the source text for the operand is located. Scan pointers are positioned inside the first encountered left parenthesis. At this point processing varies according to the type of address constant: A and Y, Q, V and R, or S.

For A and Y types, EVAL is called and the results of the evaluation are analyzed. Absolute values are placed directly in the text. These may be bit-oriented fields. Literals and erroneous expressions are diagnosed and rejected. Relocatable and complex expressions are presented to PUTVAL for construction of the text and relocation information, unless the field is a bit-oriented one. Bit-oriented fields are rejected for relocatable values, diagnosed, and set to zero.

For Q, V and R types, the characters of the symbol are collected, and, for V and R types, the matching V-type entry in the permanent dictionary is located. For V and R types, values are extracted from the item, and PUTVAL is called to construct the text and relocation information. For Q types, the operand is tested for validity and a cross reference item is built if requested. A call is then made to PUTVAL.

For S types, EVAL is called and the results of the evaluation are separated into absolute and relocatable. Other results are diagnosed and rejected. USEVAL is then called to provide a base register and displacement that are equivalent to the original expression.

For all types, delimiters are checked and multiple constants are processed until

the terminating right parenthesis is reached.

Error Checks:

- Invalid expression type.
- Bit-length field for relocatable value.
- Literal appears in address constants.
- Field improperly delimited.
- Register value too large.
- Displacement value too large.
- Truncated value.
- Relocatable adcon occurs in a PUBLIC control section.
- Improper operand for Q-type adcon.

LITXT -- Phase III Literal Pooling Processor (CEVLT)

This routine places the binary text of the literals in the specified pool into the output module. The values of address constants not previously obtained are resolved during this processing. (See Chart CT.)

Entry Point: CEVLT

Calling Sequence: INVOKE ALITXT

Routines Called: DCTXT, LIST

Exit: Normal

OPERATION: This routine is called during Phase III when the current logical order file entry indicates that a LTOrg statement is to be processed. A dummy LTOrg is inserted at the end of the first control section for all literals not otherwise pooled.

The logical order file points to the head of a chain connecting all literals pooled under the given LTOrg. POOLIT has assigned location counter values to the literals on the basis of their length, and has reordered the pool chain so that all literals in the pool occur on the chain in ascending location counter order. The reordering is for the benefit of an orderly listing during Phase III.

LTOrg processing during Phase III proceeds through the pool chain. For each literal in the pool, an artificial source line is created for the benefit of the listing. This line is prefixed by the standard control bytes, and contains blanks

in the first 15 columns; the = is placed in column 16, followed by the source text.

An artificial logical order file entry is created for the literal. The logical order file entry simulates that of a normal DC statement, except that the multiple operand flag bit is always off, since multiple operands are prohibited on literals. The simulated logical order file entry is chained to itself. Having made the literal appear as if it were a normal constant, DCTXT is called to process the constant.

If the literal is an address constant referring to the location counter, the first trailer that is not flagged as generated is found, the pseudo dictionary item for the location counter is set to the value contained in the trailer, and the trailer is flagged as processed. Any address constant produced by DCTXT will contain the location counter value of the literal reference statement when asterisk notation is used in the expression.

CSDPR -- CSD Processor (CEVCD)

This routine completes the processing of a control section dictionary before the Phase III control module (CEVPE) begins processing a new control section. It is responsible for retrieving all relocation modifiers and reference items in temporary storage and for producing a final output control section dictionary from them. (See Chart CU.)

Entry Point: CEVCD

Calling Sequence: INVOKE ACS DPR

Routines Called: PUTVAL, VMFREE

Exit: Normal.

OPERATION: This routine initially constructs a section heading in the output PMD. The total number of bytes in the text and the relative page number where the text begins are placed in the section heading.

The section name is located in the dictionary; entry names have been attached to section names so the associations necessary for R-type references are always properly defined. This routine follows the chain of ENTRY names attached to the section name item. The first pass over the ENTRY chain pulls off all entry names with simply relocatable definitions. A definition is simply relocatable if the value of the definition (shown in the value item) is simply relocatable, and the section number in the value item is the same as the current section. If these conditions are met, a definition item is constructed in the con-

trol module, the displacement portion of the relocatable value is entered as the value of the definition, and the alphameric name of the entry is copied into the definition term. The entry is removed from the chain.

When the end of the ENTRY chain is reached, the remaining links are rescanned for definitions with absolute values. A definition is absolute if the value of the definition as shown in the value item is absolute. Each absolute definition is entered into the output module and removed from the chain.

The remaining links of the ENTRY chain now represent complex definitions. A definition with R-type referenceability is complex because the section number in the value item is not the same as the current section. The value of such a definition can be either simply or complexly relocatable, regardless of its R-type referenceability.

Thus complex definitions may be divided into two types: simply relocatable definitions which have become complex by virtue of their R-type reference attribute; and definitions that were complexly relocatable to begin with, and that may or may not have an R-type attribute.

For the first type, the displacement portion of the relocatable value becomes the value of the definition item. The control section in which the symbol is defined is indicated by constructing an RLD modifier for the definition. The byte address of the modifier points to the value part of the definition; the type field (T) indicates addition; and the reference number points to a reference item for the name of the remote section in which the entry point is defined.

For the second type, the value item contains a quasi RLD string (originating from EVAL which indicates the names of the relocatable symbols and the operations (addition or subtraction) to be performed on them. This string is transformed into a series of RLD modifiers for the complex definition; this is the same transformation that produced the RLD modifiers for the text.

RLD modifiers for complex definitions are stacked in temporary storage in the same way the RLD modifiers for the text were. When all complex definition items have been constructed, the reference items are retrieved from temporary storage and copied into the output module in order of occurrence. Total counts of each type of definition and for all the references are

posted in the section heading as they become known.

If DXD statements are being processed, Q-REF entries will be made and a count maintained. A CXD-REF modifier will also be built for each CXD present.

The chain of RLD modifiers for complex definitions is now repeatedly scanned to accumulate the total number of definition modifiers on each page of the module. Normally the module contains only one page of definitions. As each count is determined it is posted in the appropriate word of the RLD. When counts have been determined for all pages, the RLD modifiers for complex definitions are retrieved in page order and copied into the module. The location of the first modifier for each page is posted with the count for that page.

When the RLD for complex definitions is complete, similar processing is performed for the RLD modifiers for the text. These modifiers are chained in working storage according to whether the reference is external or internal. The external modifiers are grouped by page, counted, and copied into the module first. Then the internal modifiers are processed.

Finally, a table is constructed in the module with one entry for each page of virtual storage represented by the text. The entry contains a pointer to the page of text. An empty page of virtual storage is indicated by a word of all 1 bits.

At this point the number of bytes in the control section dictionary is known and posted in the heading. The output module for the section is then complete.

SECTION 10: PHASE IV

INTRODUCTION

Phase IV produces the output selected by the user. Any of the following may be produced during this phase:

- Symbol table listing or cross-reference listing (both cannot be selected).
- Program module dictionary listing.
- Internal symbol dictionary.
- Internal symbol dictionary listing.

The request for an internal symbol dictionary listing is honored only if the internal symbol dictionary has also been requested.

Figure 23 illustrates the routine relationships for this phase.

Table 9 is a decision table listing the criteria for entering each Phase IV routine.

ROUTINES

PHASE IV -- Phase IV Control (CEVPF)

This routine calls the postprocessors required to produce the output options selected by the user. (See Chart CV.)

Entry Point: CEVPFX

Calling Sequence: L R15,ACEVPF
BR R15

Routines Called: ISDPR STED
ISDSA XREF
PMDLS

Exit: To AC (CEVAC)

OPERATION: The postprocessors produce the symbol-table listing, cross-reference listing, program module dictionary (PMD) listing, internal symbol dictionary (ISD), and ISD listing. Any combination of these services is available to the programmer. In Phase IV the option flag for each processor is checked, and the postprocessor is called if its output is desired. After the options are checked, an exit is made to the assembler main control.

If an ISD has not been requested and an ISD listing has been requested, a request for the latter is ignored.

XREF -- Cross-Reference Listing Processor (CEVXF)

This routine sorts the cross-reference items produced during Phase III and produces an orderly listing of them. (See Chart CW.)

Entry Point: CEVXF

Calling Sequence: INVOKE AXREF

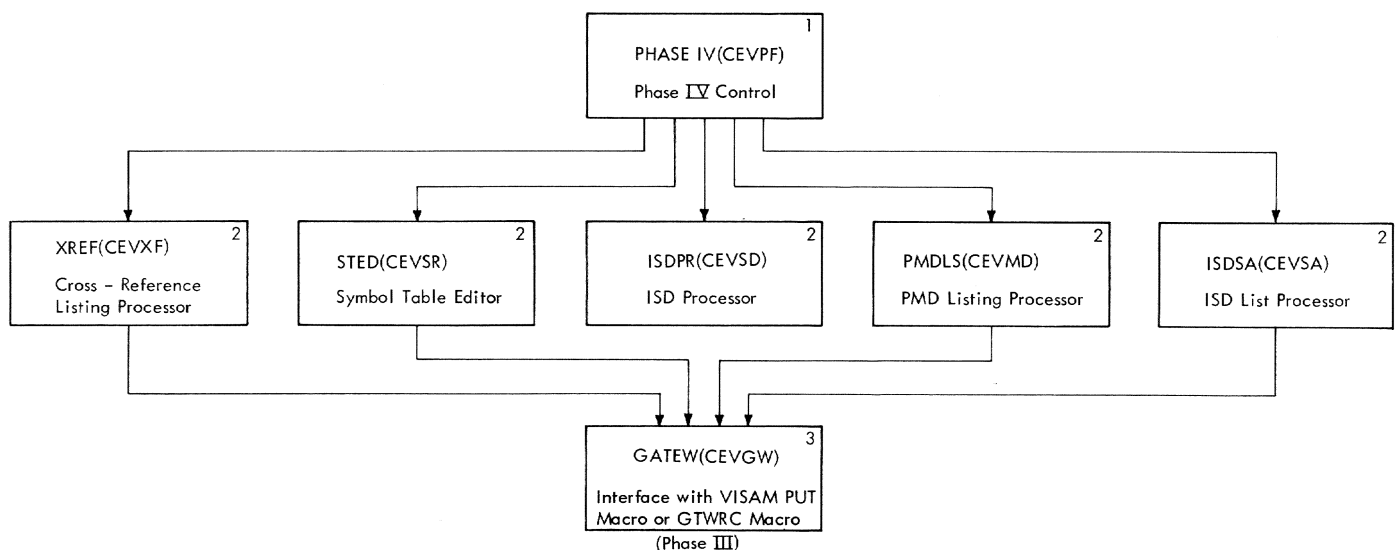


Figure 23. Phase IIV routine relationships

Table 9. Phase IV decision table

Routine: Phase IV Control Level: 1			
Routine	Purpose	Called Routines	Calling Conditions
PHASE IV (CEVPF)	Calls the post processor modules to produce the output options selected by the user.	XREF -- CEVXF	Cross-reference listing requested.
		STED -- CEVSR	Symbol table listing requested but not cross-reference listing.
		ISDPR -- CEVSD	Internal symbol dictionary requested.
		PMDLS -- CEVMD	A program module dictionary listing requested.
		ISDSA -- CEVSA	Both internal symbol dictionary and internal symbol dictionary listing requested.
Routine: Level: 2			
ISDPR (CEVSD)	Reduces the content of the main dictionary to a special internal symbol dictionary used by program checkout subsystem.	VMGET VMFREE	Get virtual storage pages for ISD. Free excess storage obtained.
ISDSA (CEVSA)	Displays the contents of the internal symbol dictionary.	GATEW -- CEVWG (Phase III)	The following are present: • carriage control character • location print line image • VISAM index • location data control block
PMDLS (CEVMD)	Prepares a listing of the program module dictionary.	GATEW -- CEVWG (Phase III)	Always called.
STED (CEVSR)	Prepare edited symbol table listing.	GATEW -- CEVWG (Phase III)	Always called.
XREF (CEVXF)	Prepare cross-reference listing.	GATEW -- CEVWG (Phase III)	Always called.

Routines Called: GATEW

Exit: Normal

OPERATION: Production of the cross-reference listing is indicated by a parameter to the assembly which is passed for the user through the calling sequence.

The logical order file is scanned during Phase III. The logical order file indicates when a symbol appears in the name field of a statement. If the cross-reference option is elected when a symbol is found, a cross-reference definition item is constructed and placed in temporary storage. In addition, a definition item is created for each external name. Cross-reference items are stacked contiguously in

the area formerly used by the macro dictionaries and do not require linkage. A cross-reference definition item has the format shown in Figure 24.

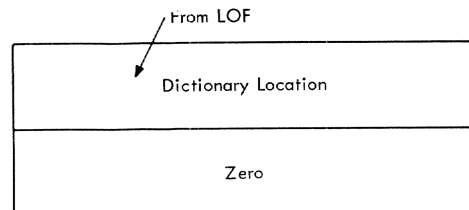


Figure 24. Cross-reference definition format

In Phase III, during operand field scanning of appropriate statements (those that produce pertinent cross-references), the cross-reference list option is tested by EVAL. In this mode, for every symbol whose lookup produces a satisfactory result, a cross-reference reference item is constructed and placed in temporary storage. This technique insures that symbols referenced within expressions appear on the listing, regardless of the ultimate relocation properties of the expression. A reference item has the format shown in Figure 25.

A separate pass is taken to produce the listing. The items are sorted alphabetically by key of dictionary item, with definitions preceding references, and references sorted by ascending value of location counter. A simple internal merge sort is used to order the items. The address list produced by the sort controls the order of the printed items. The formatted lines are stacked behind the listing in the listing area.

STED -- Symbol Table Editor (CEVSR)

This routine prepares a sorted listing of all ordinary symbols contained in the main dictionary, together with their type, length, and value attributes. (See Chart DA.)

Entry Point: CEVSR

Calling Sequence: INVOKE ACEVSR

Routines Called: GATEW

Exit: Normal

OPERATION: This routine is called during Phase IV if the symbol table listing option has been taken, and the cross-reference listing option has not also been specified. Each link indicated by the main hashing table is followed, and a sorting key consisting of the address of each item in the main dictionary, except transitive items, blank names, and variable symbols, is stored in working storage 1. The keys are then sorted into ascending alphameric sequence based on the character value of the symbols in the dictionary.

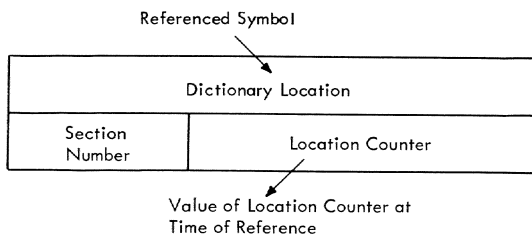


Figure 25. Reference item format

The resulting list is then edited for printing, with two columns of symbols appearing on each page. Either the VISAM PUT macro is used to place the edited lines into the list data set, or the GTWRC macro is used to place the line immediately on SYSOUT, depending on user request.

ISDPR -- ISD Processor (CEVSD)

This routine creates the internal symbol dictionary (ISD), used by the program control system. (See Chart DB.)

Entry Point: CEVSD

Calling Sequence: INVOKE AISDPR

Routines Called: VMGET, VMFREE

Exit: Normal

OPERATION: This routine is called by the Phase IV control module (CEVPF) to reduce the contents of the main dictionary to those items required by the program control system, and to format those items to create the internal symbol dictionary (ISD).

The ISD consists of three parts: a section name table; a collection of using register tables; and a collection of symbol entries.

Section Name Table: The section name table is placed in the ISD by following the chain of section names through the dictionary. The name of blank common is transferred as binary zeros. The order corresponds to the numbering order previously established for section numbers so that location counter references in the dictionary do not have to be adjusted for debugging output.

Using-Register Tables: If, during Phase IIC, the ISD output is on, an independent chain of pointers to the using-register tables was constructed for the ISD processor. ISDPR follows this list and moves each using table from its original location into the ISD. Phase III inserts the location counter value into the first word of the predefined table at the time it starts to use each such table.

When the tables are transferred to the debugging output, those entries containing external base values are marked as unused; also, the position of the section number is moved to a more convenient position within each entry.

Note that since Phase III operates in ascending location order within ascending section numbers, the list of using tables will be in ascending location order, as required by PCS.

Symbol Entries: Each link of the hash table is followed in order to inspect all the symbols in the dictionary. For each absolute or simply relocatable value item, and for each section name, an appropriate address pointer is constructed in the working storage area previously used by the cross-reference items. Symbols appearing in DSECTs are included and designated as such.

When all symbols have been extracted, the address list is sorted in such a way that the symbols are ordered by ascending location counter value, with all absolute EQUs appearing at the end, arbitrarily sorted by value.

The symbols are transferred to the debugging output. For assembler symbols, the number of replication factors is never more than 1. The replication factor has a value of 1 for everything except EQUs and DC or DS statements. For EQUs it is zero, and for DC or DS, whatever the duplication modifier was.

Type of field is determined as follows: instructions, section names and absolute (immediate) values are identified as such. Constant types F and H are integer; E and D are real; A, V, R, and Y are address; S and X are logical; and C is character string. The assembler has nothing equivalent to complex. (This is type of number, not relocation property.) Constant type B is logical, P is logical, and Z is character string.

Counts of the number of section names, using tables, and symbol entries are maintained and posted at the head of the debugging output.

PMDLS -- Program Module Dictionary Listing Processor (CEVMD)

This routine prepares a listing of the program module dictionary. (See Chart DC.)

Entry Point: CEVMD

Calling Sequences:

```
L      R15,APMDLS
CALL   (15),MF,E,xxxx
        xxxx address of a four-
        word parameter list.
```

Input Parameters:

R1 Address of a four-word list containing the following information:

Word 1 - Location of program module dictionary

Word 2 - Length of program module dictionary
Word 3 - Last page number assigned in assembler output
Word 4 - Address of 550-byte word area (must be on double-word boundary)

Routines Called: GATEW

Exit: Normal

OPERATION: Information for the listing header lines is secured from the program module dictionary header. The following details, when present, are listed for each control section within the module:

- Section name.
- Type of section.
- Time stamp.
- Attributes.
- Length of the control section.
- Text length.
- Relocatable, absolute, and complex definitions for the section.
- References.
- DXD and CXD references.
- Modifiers for complex definitions.
- Modifiers for text (internal and external references, Q-CONS, and CXDs).

ISDSA -- ISD List Processor (CEVSA)

This module displays the contents of the internal symbol dictionary (ISD). (See Chart DD.)

Entry Point: CEVSA

Calling Sequence: INVOKE AISDLS

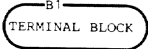
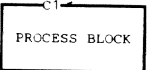
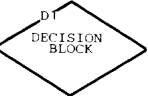
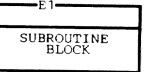
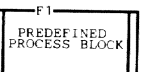
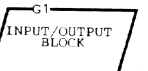
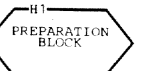


Routines Called: GATEW

Exit: Normal

OPERATION: The ISD list consists of five line groupings. Each line may contain up to twelve symbol names, types, duplication factors, symbol lengths, or locations, and/or values for all symbol entries in the ISD.

SECTION 11: FLOWCHARTS

The flowcharts in this manual have been produced by an IBM program, using ANSI symbols. The symbols are defined in the left column below, and examples of their use are shown at the right.

SYMBOL	DEFINITION	EXAMPLE	COMMENTS
	INDICATES AN ENTRY OR TERMINAL POINT IN A FLOWCHART; SHOWS START, STOP, HALT, DELAY OR INTERRUPTION. MAY ALSO INDICATE RETURN TO THE CALLING PROGRAM.	MODNAME B3 COMNAME	B3: MODNAME IS THE LOAD MODULE OR LIBRARY NAME OF THE ROUTINE DESCRIBED BY THIS FLOWCHART. COMNAME IS THE COMMON NAME OF THE ROUTINE.
	INDICATES A PROCESSING FUNCTION OR A DEFINED OPERATION CAUSING CHANGE IN VALUE, FORM OR LOCATION OF INFORMATION.	FROM: OTHERMOD CHART AZ C3 CSECT LABEL1	OTHERMOD INDICATES THE MODULES PASSING CONTROL TO THIS MODULE AND THEIR FLOWCHARTS. C3: CSECT IS THE CSECT NAME OR OTHER ENTRY POINT AT WHICH PROCESSING BEGINS. LABEL1 IS THE LABEL OF THE FIRST INSTRUCTION.
	INDICATES A DECISION OR SWITCHING-TYPE OPERATION THAT DETERMINES WHICH OF A NUMBER OF ALTERNATE PATHS SHOULD BE FOLLOWED.	D3 NO YES H3	D3: PROGRAM EXECUTION CONTINUES WITH BLOCK H3 WHEN THE DECISION IS NO OR BLOCK E3 WHEN THE DECISION IS YES.
	INDICATES A SUBROUTINE OR MODULE THAT IS DESCRIBED IN THIS MANUAL.	LABEL2 E3 SUBRTN AG ENTRYPT VIA: PASSMECH	E3: LABEL2 IS THE LABEL OF THE SECTION OF CODE IN THIS ROUTINE FROM WHICH CONTROL IS PASSED TO THE SUBROUTINE. CONTROL RETURNS TO THE NEXT INSTRUCTION FOLLOWING THE SUBROUTINE CALL. ENTRYPT IS THE ENTRY POINT. SUBRTN IS THE COMMON NAME OF THE SUBROUTINE IN FLOWCHART AG. VIA: PASSMECH INDICATES HOW CONTROL PASSES FROM COMNAME TO SUBRTN.
	INDICATES A SUBROUTINE OR MODULE THAT IS INCLUDED IN THE FLOWCHARTS OF ANOTHER MANUAL.	LABEL3 F3 -PDPNM-	F3: LABEL3 IS THE LABEL OF THE SECTION OF CODE FROM WHICH CONTROL IS PASSED TO THE PREDEFINED PROCESS PDPNM, WHICH IS DOCUMENTED IN ANOTHER PUBLICATION (-PDPNM- MAY ALSO BE USED IN A PROCESSING BLOCK).
	INDICATES GENERAL I/O FUNCTIONS, SUCH AS GET, PUT, READ, WRITE, SIO, AND DEVICE CONTROL MACRO INSTRUCTIONS.	G3 NO YES 01 H3 02 A1	G3: EXECUTION CONTINUES WITH BLOCK H3 WHEN THE DECISION IS YES OR WITH BLOCK A1 ON PAGE 2 OF THIS SET OF FLOWCHARTS WHEN THE DECISION IS NO. THE OFFPAGE CONNECTOR MARKED 01H3 INDICATES THAT EXECUTION CONTINUES WITH BLOCK H3 FROM ANOTHER PAGE OF THIS SET OF FLOWCHARTS. THIS CONNECTOR IS ALSO PAIRED WITH THE ONPAGE CONNECTOR FROM BLOCK D3.
	INDICATES A PROCESS THAT CHANGES SYSTEM OPERATION FOR EXAMPLE, SETS A SWITCH, MODIFIES AN INDEX REGISTER, OR INITIALIZES A ROUTINE.	LABEL4 H3	H3: LABEL4 IS THE LABEL OF A SECTION OF CODE OF THIS ROUTINE THAT INITIATES I/O.
	INDICATES ENTRY TO OR EXIT FROM ANOTHER BLOCK ON THE SAME FLOWCHART PAGE.	J3 NEXTRTN	J3: NEXTRTN IS THE COMMON NAME OF THE ROUTINE THAT EXECUTES AFTER THIS ROUTINE. ENTRYPT IS THE ENTRY POINT OF NEXTRTN, WHICH IS DESCRIBED IN CHART AC.
	INDICATES ENTRY TO OR EXIT FROM A BLOCK ON ANOTHER PAGE OF THE SAME SET OF FLOWCHARTS.	EP=ENTRYPT CHART AC VIA: PASSMECH	VIA: PASSMECH INDICATES HOW CONTROL PASSES FROM COMNAME TO NEXTRTN.

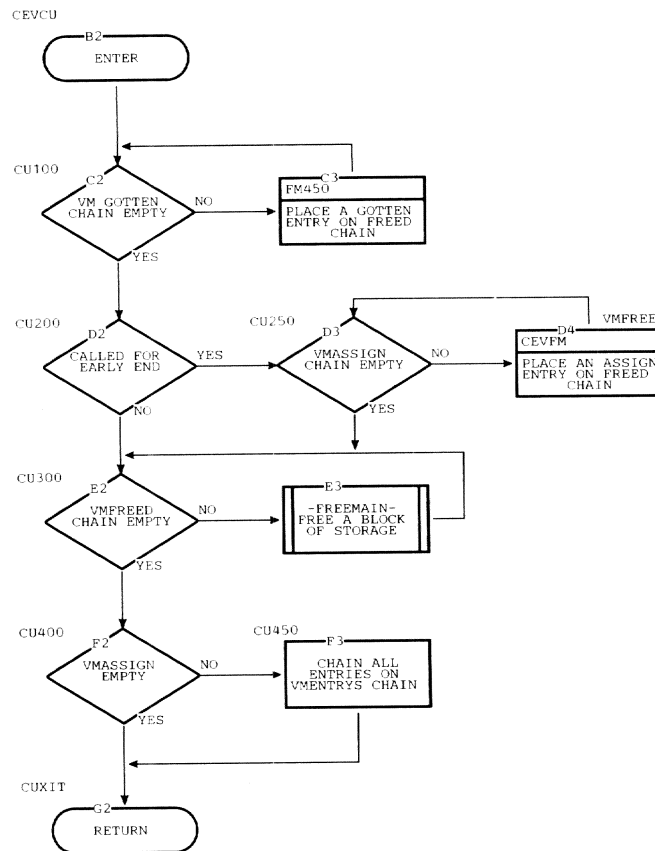
Program Logic Manual

GY28-2021-2

Assembler

Flowcharts on pages 117-258 were not scanned.

Chart EC. VMCLEAN (assembler cleanup) - CEVCU



SECTION 12: TABLES, TABLE ENTRIES, LISTING FORMATS

This section discusses the following:

- Main Dictionary.
- Logical Order File (LOF).
- Global Section Macro Chain (GSM).
- Macro Name Dictionary.
- Operation Code Table.
- Machine Operations Requirements Table.
- Using Register Tables.
- Macro Level Dictionary (Temporary Dictionary).
- Source Line Storage Control.
- Pseudo Dictionary Item for Current Location Counter.
- Constant Item Format.
- Source Program Listing.
- Symbol Table Listing.
- Cross-Reference Listing.
- Internal Symbol Dictionary (ISD) Listing.
- Program Module Dictionary Listing.
- Object Program Listing.
- Internal Symbol Dictionary (ISD).
- Program Module Dictionary (PMD).

MAIN DICTIONARY

The main dictionary is used to establish and develop the definitions for symbols in the source program. It is used by all modules that contribute to, or require information about, symbol definition.

In addition to the main dictionary, a temporary dictionary is maintained for each macro level. The temporary dictionary contains the system variable symbols, macro parameters, and sequence symbols defined at the current macro level. The main dictionary contains all names representing location counter values, absolute values, and global variable symbols; i.e., all sym-

bols whose influence is not confined to a particular macro level.

The main dictionary is a disjointed collection of variable-length entries dynamically constructed by the assembler in segment 2. The dictionary items are made accessible for reference through a hashing algorithm applied to the alphameric value of the name. Hashing produces an index value to a table. Each item in the table is the address of the most recently constructed dictionary item related to an alphameric name value.

Basic Format

The first two words of all dictionary items specify the symbol name. The third word is an identifying type and the location of the previous item that hashed to the same value. Succeeding words vary with the kind and amount of information that must be related to the symbol being defined. All dictionary items occupy an integral number of full words.

The main dictionary contains the following types of items. The entry under the type code column is the code found in the first byte of word three for each item. These items are described separately following this list.

<u>Item</u>	<u>Type Code</u>
Absolute value	00000001
Relocatable value (prepared for DS, DC, or CXD)	11000110
Relocatable value (prepared for a machine instruction, or CCW, LTOrg, or EQU statement)	11000010
DXD value	00110001
Complex value	11000011
External name (explicit)	11000100
External name (implicit)	00000000
Control section	11001000
Literal	11110000
Transitive	01000001
*Local variable symbol (LCLA)	00010001
*Local variable symbol (LCLB)	00010011
*Local variable symbol (LCLC)	00010101
Global variable symbol (GBLA)	00010000
Global variable symbol (GBLB)	00010010
Global variable symbol (GBLC)	00010100
*Sequence Symbol	10010000

*The formats for these items are described in the macro level dictionary.

Absolute Value Item

This item results from processing an EQU statement whose operand expression yielded an absolute result. It is constructed during Phases I or IIA by the EQU instruction scan module (CEVQU) when the expression is evaluatable. It is constructed during Phase IIB by the assign value to name module (CEVEQ) when the expression is relocatable or indeterminate.

The format of an absolute value item is shown in Figure 26.

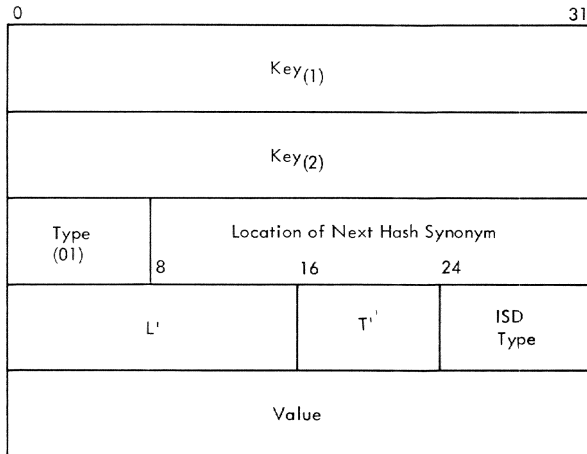


Figure 26. Absolute value item (EQU)

Words 1 and 2

8-character alphanumeric key.

Word 3, byte 1

type code - hexadecimal 01.

Word 3, bytes 2-4

location of next item whose key hashes to the same value (carried as 24-bit increment to the base of working segment 2).

Word 4, bytes 1 and 2

length attribute - an absolute integer expression with a value from 1 to 65535 (hex FFFF), or a 1-2 byte self-defining term (hex, character, or binary). If the length operand is omitted, a value of 1 is entered.

Word 4, byte 3

type attribute - an absolute integer expression with a value from 0 to 255 (hex FF), or a 1 byte self-defining term (hex, character, or binary). The reader is referred to the description of the constant item format for the meaning of the character attribute codes. If the type operand is omitted, and the symbol previously occurred in the name field of a macro reference, the type attribute is C'M'

(hex D4). The type attribute for an omitted type operand is C'U' (hex E4).

Word 4, byte 4

ISD type - type attribute to be used in Phase IV to set ISD type. This byte is filled in, but is actually not used by Phase IV. The field is introduced to be compatible with the same field in the relocatable value item entry (type X'C2').

Word 5

absolute value of symbol, carried as 32-bit integer.

Relocatable Value Item

This item is constructed by either of the following:

- The define symbol module (CEVSY), which is called when a location symbol is encountered in the name field of a machine instruction, or a DS, DC, CCW, CXD, or LTOrg statement.
- The assign value to name module (CEVEQ) in response to an EQU statement containing a simple relocatable expression.

A relocatable value item may be in one of two types of formats.

A relocatable value item prepared for a DS or DC statement has the format shown in Figure 27.

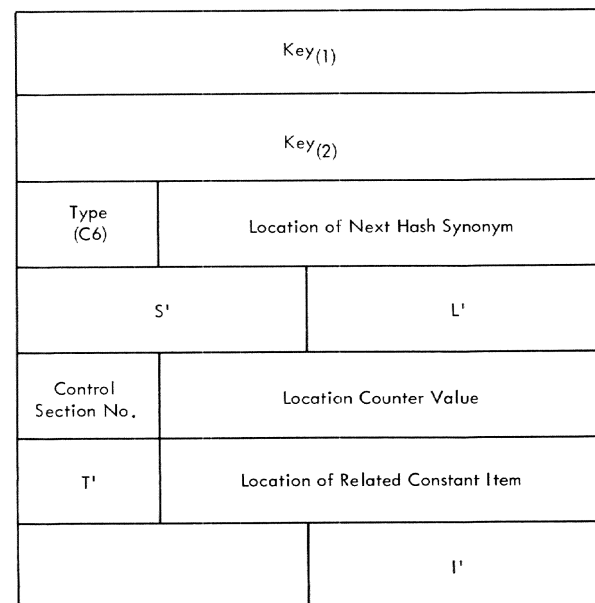


Figure 27. Relocatable value item (DC, DS, CXD)

- Words 1 and 2
8-character alphameric key.
- Word 3, byte 1
type code - hexadecimal C6.
- Word 3, bytes 2-4
location of next item whose key hashes to the same value (carried as 24-bit increment to the base of working segment 2).
- Word 4, bytes 1 and 2
scaling attribute - applicable to F, E, H, and D constants.
- Word 4, bytes 3 and 4
length attribute.
- Word 5, byte 1
number of control section in which symbol is defined.
- Word 5, bytes 2-4
displacement of symbol from base of control section.
- Word 6, byte 1
type attribute - one of the alphameric characters A, B, C, D, E, F, G, H, K, P, O, Q, R, S, V, X, Y, or Z. The reader is referred to the description of the constant item format for the meaning of the attribute codes. If the symbol previously occurred on a macro reference, the normal type attribute for the type constant will be overridden and set to the character M.
- Word 6, bytes 2-4
location of constant item containing value of constant, expressed as 24-bit increment to the base of working segment 2.

- Word 7, bytes 1 and 2
not used.
- Word 7, bytes 3 and 4
integer attribute - applicable to F, E, H, and D constants.

A relocatable value item prepared for a machine instruction, or a CCW, LTORG, or EQU statement has the format shown in Figure 28.

- Words 1 and 2
8-character alphameric key.
- Word 3, byte 1
type code - hexadecimal C2.
- Word 3, bytes 2-4
location of next item whose key hashes to the same value, expressed as 24-bit increment to the base of working segment 2.

- Word 4, bytes 1 and 2
length attribute - an absolute integer expression with a value from 1 to 65535 (hex FFFF), or a 1-2 byte self-defining term (hex, character, or binary). If the length operand is omitted, the following values will be entered: 8 (CCW), 6, 2, or 4 (machine instruction), 1 (LTORG), or variable (EQU).

- Word 4, byte 3
type attribute - an absolute integer expression with a value from 0 to 255 (hex FF), or a 1 byte self-defining term (hex, character, or binary). If the type operand is omitted, and the symbol previously occurred in the name field of a macro reference, the type attribute is C'M' (hex D4). The type attribute for an omitted type operand is C'U' (hex E4).

- Word 4, byte 4
ISD type - type attribute used by Phase IV to set ISD code for the symbol. This field has an ISD type attribute identical to byte 3 when an EQU type operand is specified. This byte will differ from byte 3 when it is preferable to have an ISD code assigned which is different from that which the type attribute implies.

- Word 5, byte 1
number of control section in which symbol is defined.

- Word 5, bytes 2-4
displacement of symbol from base of control section.

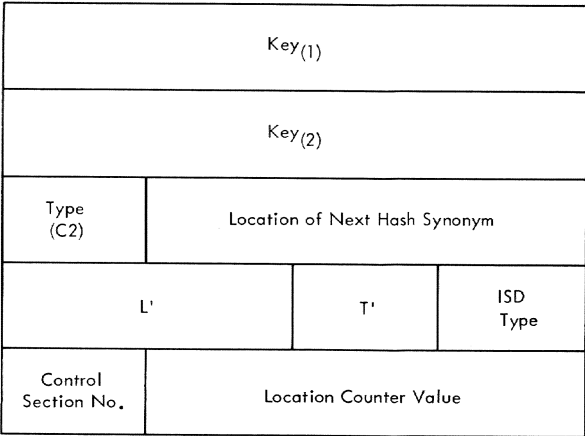


Figure 28. Relocatable value item (machine instructions, CCW, LTORG, EQU)

DXD Item

This item is constructed by the define symbol routine (CEVSY) which is always called by the CXD Phase I Processor (CEVCX).

The dictionary item prepared for a DXD statement has the format shown in Figure 29.

Key (1)	
Key (2)	
Type (31)	Location of Next Hash Synonym
External Reference No.	L'
Location of Next External Name	
T'	Location of Related Constant Item
Not Used	I'

Figure 29. DXD item

Words 1 and 2

8-character alphameric key.

Word 3, byte 1

type code - hexadecimal 31.

Word 3, bytes 2-4

location of next item whose key hashes to the same value (carried as 24-bit increment to the base of working segment 2).

Word 4, bytes 1 and 2

number of this symbol in the external reference table produced for the control section dictionary for the current control section during Phase III. This number is cleared to all bits X'FFFF' between control sections and reassigned the first time the symbol is referenced within a given section.

Word 4, bytes 3 and 4

length attribute.

Word 5

location (32-bit) of next external name item (either explicit or implicit) defined within the assembly.

Word 6, byte 1

type attribute - one of the alphameric characters.

Word 6, bytes 2-4

location of constant item containing value of constant, expressed as 24-bit increment to the base of working segment 2.

Word 7, bytes 1 and 2

not used.

Word 7, bytes 3 and 4

integer attribute - applicable to F,E,H, and D constants.

Complex Value Item

This item is constructed by the EQU instruction scan during Phases I and IIA and the assign value to name routine (CEVEQ) during Phase IIB, when an EQU statement with an expression referencing multiple relocatable symbols or a single external symbol is processed. The complex value is represented by the string of terms output by the expression evaluator routine (CEVEV) to describe the simplified expression.

The format of a complex value item is shown in Figure 30.

Key(1)		
Key(2)		
Type (C3)	Location of Next Hash Synonym	
L'	T'	Not Used
*	Not Used	No. of Bytes in Relocatable Value String
+ -	Absolute Value	
+ -	Operator Code	Location of Dictionary Item for Relocatable, or External Term n
Operator Code		
Location of Dictionary Item for Relocatable, or External Term n		

Figure 30. Complex value item (EQU)

Words 1 and 2

8-character alphameric key.

Word 3, byte 1

type code - hexadecimal C3.

Word 3, bytes 2-4

location of next item whose key hashes to same value, expressed as 24-bit increment to the base of working segment 2.

Word 4, bytes 1 and 2

length attribute - an absolute integer expression with a value from 1 to 65535 (hex FFFF), or a 1-2 byte self-defining term (hex, character, or binary).

Word 4, byte 3
 type attribute - an absolute integer expression with a value from 0 to 255 (hex FF), or a 1 byte self-defining term (hex, character, or binary).

Word 4, byte 4
 not used.

Word 5, byte 1
 first bit on indicates symbol was equated to an external symbol and that the symbol is really not complex.

Word 5, bytes 2 and 3
 not used.

Word 5, byte 4
 number of bytes in the list of relocatable values (words 6 - n).

Word 6
 absolute value portion of complex expression, carried as 32-bit signed integer.

Words 7 through n:

byte 1
 operator code to be applied to the relocatable term. The only allowable operators are addition and subtraction. The hexadecimal codes are F6 and F4 respectively.

bytes 2-4
 location of dictionary item for relocatable symbol, expressed as 24-bit increment to the base of working segment 2.

Note: The expression evaluator module (CEVEV) determines the operator codes and dictionary locations.

External Name Item

This item may be created in one of two ways:

- By EXTRN for each operand of an EXTRN statement (explicit item).
- By CSCAN for each operand of a V-type address constant (implicit item).

Both explicit and implicit items perform the same function: providing a definition point for an external reference. However, implicit items are transparent to all normal references; they are visible only when V-type constants are processed.

The format of an explicit external name item is shown in Figure 31.

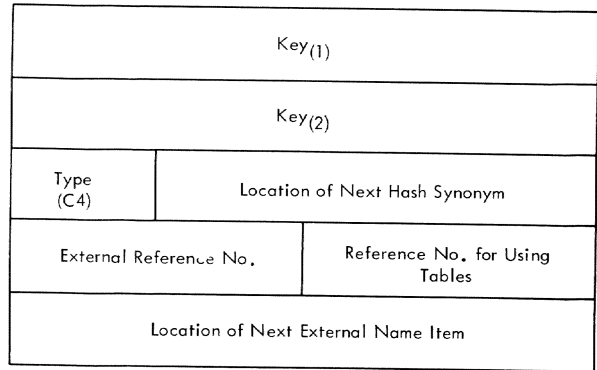


Figure 31. External name item (EXTRN)

Words 1 and 2
 8-character alphanumeric key.

Word 3, byte 1
 type code - hexadecimal C4.

Word 3, bytes 2-4
 location of next item whose key hashes to the same value expressed as a 24-bit increment to the base of working segment 2.

Word 4, bytes 1 and 2
 number of this symbol in the external reference table produced for the control section dictionary for the current control section during Phase III. This number is cleared between control sections and reassigned the first time the symbol is referenced within a given section.

Word 4, bytes 3 and 4
 sequentially assigned number for identifying external symbols which appear in using register tables. External symbols which are so used are assigned sequentially ascending reference numbers by Phase IIC.

Word 5
 location (32-bit) of next external name item (either explicit or implicit) defined within the assembly.

The format of an implicit external name item is shown in Figure 32.

Words 1 and 2
 8-character alphanumeric key.

Word 3, byte 1
 type code, hexadecimal 00.

Word 3, bytes 2-4
 location of next item whose symbol hashes to the same value, expressed as a 24-bit increment to the base of working segment 2.

Word 4, bytes 1 and 2
 number of this symbol in the external reference table produced for the control section dictionary for the current control section during Phase III. This number is cleared between control sections and reassigned the first time the symbol appears in a V-type address constant within a given section.

Note: V-type external name items are not visible to users of the normal dictionary lookup routines; they are, in effect, undefined symbols with respect to the rest of the assembly. The lookup used to process V-type address constants is the only one cognizant of this item type.

Word 4, bytes 3 and 4
 not used.

Word 5
 location (32 bits) of next external name item, either implicit or explicit, defined within the assembly.

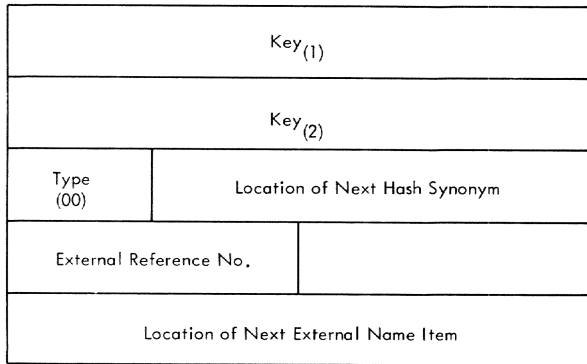


Figure 32. External name item (V-type address constant)

Control Section Item

This item is constructed by the statement analyzer module (CEVST) upon the first occurrence of a symbol (or blanks) in the name field of a CSECT, DSECT, COM, START, or PSECT statement.

This item contains two value fields:

- The current value - a running counter maintained during Phases IIB and III.
- The maximum attained value:
 1. Provides the resumption location for an ORG statement with a blank operand field during Phase IIB.
 2. Contains the object size of the control section during Phase III.

The format of a control section item is shown in Figure 33.

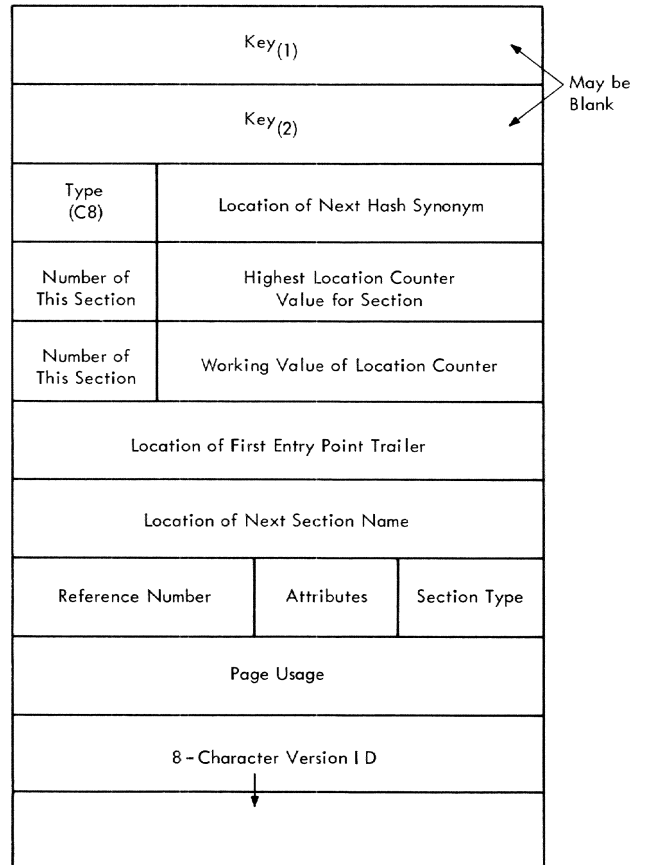


Figure 33. Control section item (CSECT, DSECT, COM, START, PSECT)

Words 1 and 2
 8-character alphameric key.

Word 3, byte 1
 type code - hexadecimal C8.

Word 3, bytes 2-4
 location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 2.

Word 4, byte 1
 number of this control section. Sections are numbered sequentially from one in order of their occurrence.

Word 4, bytes 2-4
 highest attained displacement reached by the location counter for this section. This value is maintained and updated by the ORG and section name routines in Phase IIB.

Word 5, byte 1
 number of this control section. Duplicated for convenience of processing this word.

Word 5, bytes 2-4
current displacement of the location counter for this section. Used by the ORG and section name routines of Phase IIB.

Word 6
location of the first entry point trailer for this control section. This is the head of a list of entry points for the section.

Word 7
location of next section name item. This is one of the entries in a list of section names maintained for the assembly as a whole.

Word 8, bytes 1 and 2
number of this symbol in the external reference table produced for the control section dictionary for the current control section during Phase III. This number is cleared to all bits between control sections and is re-assigned the first time a relocation modifier for the text is required that refers to a symbol defined within this section.

Word 8, byte 3
definition of the attributes of this section. The codes are:

PUBLIC	00001000
READONLY	00010000
VARIABLE	00100000
SYSTEM	01000000
PRVLGD	10000000

The bit configuration 00000001 indicates the control section has been processed by Phase IIA.

Word 8, byte 4
definition of the type of control section. The codes are:

COM	00000001
Blank COM	00000010
CSECT	00000011
DSECT	00000100
PSECT	00000101
Blank CSECT	00000110

Word 9
during Phase IIB this field is used to hold a cumulative count of the number of different virtual storage pages used within the assembled control section. This statistic is required in order to request sufficient working storage to produce the binary text in Phase III. The page usage for all control sections is added together during Phase IIB processing, and an appropriate GETMAIN macro is issued in Phase III.

Words 10 and 11
8-character information indicating version identification for the control section. All control section items are members of a chain. The entry trailer format is shown in Figure 34.

Entry Trailer Item

The format of an entry trailer item is shown in Figure 34.

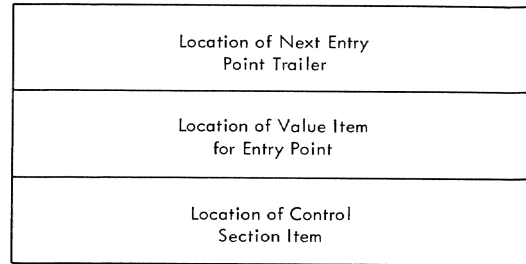


Figure 34. Entry trailer item

Word 1
location of next entry point trailer for this section (32 bits).

Word 2
location of the relocatable value item or absolute value item corresponding to the definition of the entry point (32 bits).

Word 3
location of the section name item for the section in which the symbol is defined. (This is not necessarily the item to which the trailer is attached.)

Note: An entry point trailer is attached to the section name item for each ENTRY operand declared within the section. Because of R-type constants, the section with which an ENTRY operand is associated is not always the section within which the entry symbol is defined.

Literal Item

This item is constructed during Phase IIB for each literal operand encountered. It contains the entire source text for the literal, permitting positive identification in dictionary lookup when the literal text is expressed in more than eight characters.

A literal item contains two pointers:

- A pointer to the constant item produced by the constant scan module (CEVCS).
- A pointer to the beginning of the chain of trailer items.

Trailer items ensure proper duplication when the same literal is used under different literal pool origins, and when the literal contains an address constant which refers to the value of the location counter.

The format of a literal item is shown in Figure 35.

Key(1)	
Key(2)	
Type (F0)	Location of Next Hash Synonym
L'	Length of Source
Location of Constant Item	
Location of First Trailer	
Location of Source Text	
I'	S'

Figure 35. Literal item

- Words 1 and 2
8-character alphameric key derived from the text of the literal.
- Word 3, byte 1
type code - hexadecimal F0.
- Word 3, bytes 2-4
location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 2.
- Word 4, bytes 1 and 2
length attribute.
- Word 4, bytes 3 and 4
length of source text of literal, (excluding the =) in bytes.
- Word 5
location of the constant item which contains the value of the literal (32 bits); the constant item is produced during Phase IIB by the constant scan module (CEVCS).
- Word 6
location of the first trailer for this item (32 bits); trailers are discussed below.

- Word 7
location of source text, exclusive =; 32 bits.
- Word 8, bytes 1 and 2
integer attribute - for F, E, D, and H constants.
- Word 8, bytes 3 and 4
scaling attribute - for F, E, D, and H constants.

The format of the trailer for a literal item is shown in Figure 36.

Location of Literal Item	
Flags	Next Trailer Location
LTORG No.	Next Literal in Pool
Section No.	Location Counter Value of Literal Definition
Section No.	Location Counter Value of Reference [=A (*)]

Figure 36. Literal trailer item

- Word 1
location of literal item in main dictionary (32-bits).
- Word 2, byte 1
flag bits indicating the status of literal processing:
- bits 0-5
not used.
- bit 6
if 1, indicates that a fifth word is present in the trailer, and that this literal contains a reference to the current location counter. If 0, indicates that the trailer contains only four words and does not contain a reference to the current location counter.
- bit 7
this bit is used by the lookup routine in Phase III to ensure unique results for literals that reference the location counter, as indicated by bit 6 on. When the value of such a literal is retrieved from the trailer, bit 7 is set on to prevent this value from being retrieved again on any subsequent lookup. This forces a unique

value for each reference to the location counter.

Word 2, bytes 2-4
location of next trailer item for this literal, expressed as a 24-bit increment to the base of working segment 2.

Word 3, byte 1
LTORG number controlling the reference represented by this trailer. See the description of the RESLIT subroutine in Phase IIB for a description of the LTOrg number.

Word 3, bytes 2-4
location of the next literal in the pool for this LTOrg number. This is a 24-bit increment to the base of working segment 2 and is the location of the first byte of the trailer item for this LTOrg number.

Word 4, byte 1
number of the section in which the binary text of the literal is placed.

Word 4, bytes 2-4
displacement from the base of the section at which the binary text of the literal begins.

Word 5, byte 1
number of the section represented by the asterisk notation in a reference to the current location counter.

Word 5, bytes 2-4
displacement from the base of the section represented by the asterisk notation reference to the current location counter.

Transitive Item

This item is constructed by EVAL whenever a reference is encountered to a symbol for which a definition item does not exist in the dictionary.

Whenever an EQU statement name which EQU cannot evaluate is encountered in Phases I or IIA, a transitive item is also made.

Transitive items created under the first condition are dynamically constructed in working segment 2 during Phases I and IIA and entered into an open-ended chain such that the last transitive item points to the previous item. At the end of Phase IIA, if in conversational mode, a pass is made over the transitive items to produce a diagnostic message for each undefined reference.

Transitive items created under the second condition are not entered into the transitive chain since they are incomplete definitions rather than a reference. The format of a transitive item is shown in Figure 37.

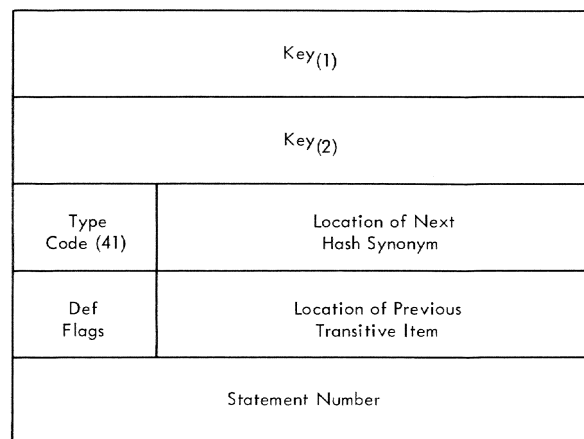


Figure 37. Transitive item

Words 1 and 2
8-character alphanumeric key.

Word 3, byte 1
type code - hexadecimal 41.

Word 3, bytes 2-4
location of next item whose symbol hashes to the same value, expressed as a 24-bit increment to the base of working segment 2.

Word 4, byte 1
definition status flag. Hex values:

00 - definition not yet received.

01 - definition received.

02 - symbol appears on a machine operation instruction operand in a DSECT.

20 - symbol appears on a macro instruction.

Transitive items whose status flag is still zero at the end of Phase IIA represent undefined symbols. If the symbol is defined, it can be found by following the hash synonym chain. If the symbol appears on a macro instruction, its type attribute is set to M at such time as the final definition is made.

Word 4, bytes 2-4
location of previous transitive item, carried as a 24-bit increment to the base of working segment 2.

Word 5
number of source statement on which symbol appeared. Statement number is carried as a 7-digit signed, packed decimal number.

Local Variable Symbol Items

Local variable symbols are those defined by the LCLA, LCLB, and LCLC statements.

These symbols are used primarily within macro definitions and are therefore described in the macro level dictionary. However, local symbols may be used outside of macros, in which case the item for them are placed in the main dictionary in the same format as used in the macro level dictionary.

Global Variable Symbol Items

Global variable symbols are defined by the GBLA, GBLB, and GBLC statements. The values of these symbols are independent of macro level; therefore, their dictionary items are maintained in the main dictionary.

Each subscripted global variable symbol item contains the maximum permissible subscript, and trailers for each non-null value, together with an indication of their respective subscripts.

Each unsubscripted global variable symbol item provides space for the setting of a non-null value.

Global items are reset to null values the first time they are reprocessed during Phase IIA so the sequence of value changes can be synchronized with the macro expansion process. The status of this resetting is carried in a flag bit in each global item. The flag bit is interrogated and reset during Phase IIA by the global/local instruction scan module (GBLX/ICLX).

There are three different formats for the global variable symbol items: global arithmetic item, global Boolean item, global character item. The format of each is described below.

The format of a subscripted global arithmetic item is shown in Figure 38.

Key(1)		
Key(2)		
Type Code (10)	Location of Next Hash Synonym	
Max SS Value	Status Flag	
	Location of First Subscript	

Figure 38. Subscripted global arithmetic item

Words 1 and 2
8-character alphameric key.

Word 3, byte 1
type code - hexadecimal 10.

Word 3, bytes 2-4
location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 2.

Word 4, byte 1
maximum allowable subscript value defined for this item (1-255).

Word 4, byte 2

bit 0
redefinition flag. If 0, the symbol has not been reprocessed by Phase IIA. In Phase IIA a zero in this bit causes the value of the item to be reset to null (zero) and the bit to be set on. The item is set to null by zeroing the subscript link or the value, as the case may be.

bits 1-7
not used.

Word 4, bytes 3 and 4
not used, for ease of processing GBL and LCL symbols together.

Word 5
location of the first subscript trailer, expressed as a 24-bit increment to the base of working segment 2.

The format of the subscript trailer is shown in Figure 39.

This SS Value	Location of Next Subscript
Value	

Figure 39. Subscript trailer for subscripted global arithmetic item

Subscript trailers:

Word 1, byte 1
value of the subscript represented by the trailer. Trailers are present only for those subscripts which are not null.

Word 1, bytes 2-4
location of next subscript trailer, expressed as a 24-bit increment to the base of working segment 2.

Word 2
32-bit signed value of the arithmetic symbol.

The format of an unsubscripted global arithmetic item is shown in Figure 40.

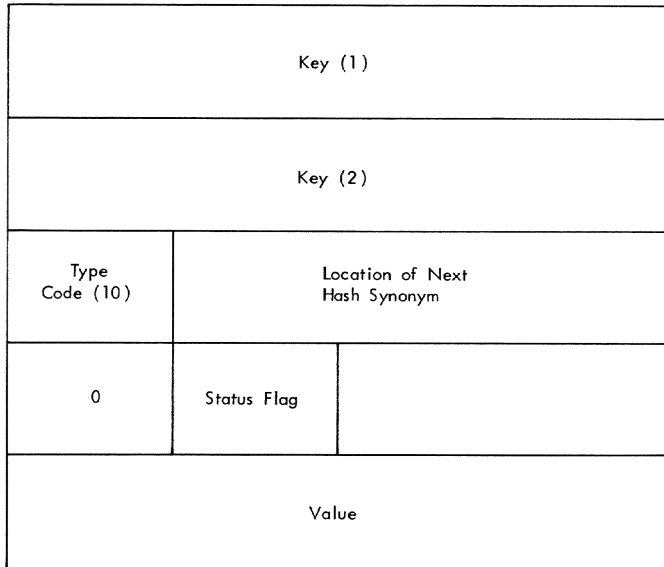


Figure 40. Unsubscripted global arithmetic item

The words have the same meaning as when the item is subscripted, except:

Word 4, byte 1
zero.

Word 5
32-bit signed value of the arithmetic symbol.

The format of a subscripted global Boolean item is shown in Figure 41.

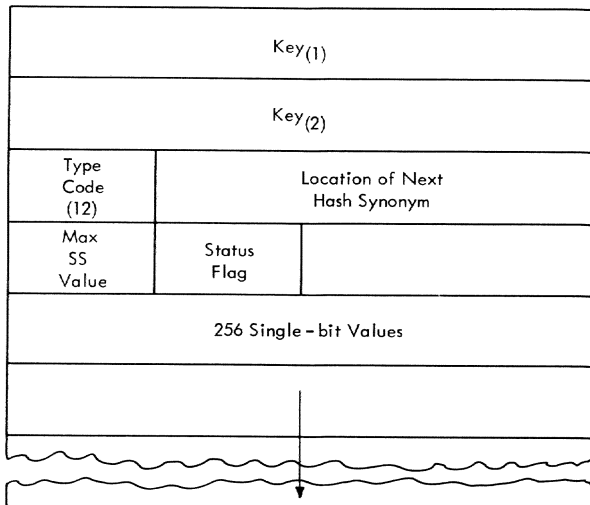


Figure 41. Subscripted global boolean item

Words 1 and 2
8-character alphaneric key.

Word 3, byte 1
type code - hexadecimal 12.

Word 3, bytes 2-4
location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 2.

Word 4, byte 1
maximum allowable subscript value defined for this item (1-255).

Word 4, byte 2

bit 0
redefinition flag. If 0, the symbol has not been reprocessed by Phase IIA. In Phase IIA a zero in this bit causes the value of the item to be reset to null (false) and the bit to be set on. The item is set to null by clearing the eight subscript words, or the value bit, as the case may be.

bits 1-7
not used.

Word 4, bytes 3 and 4
not used, for ease of processing GBL and LCL symbols together.

Words 5-12
256 single-bit values. The first bit represents subscript 0 and is not used. The remainder represent subscripts 1-255 and represent true when 1 and false when 0.

The format of an unsubscripted global Boolean item is shown in Figure 42.

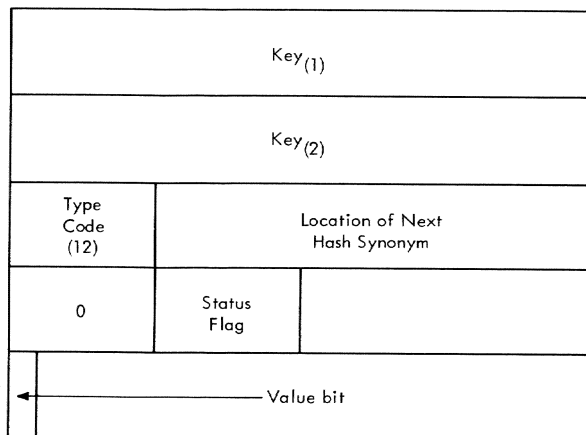


Figure 42. Unsubscripted global boolean item

The words have the same meaning as when the item is subscripted, except:

Word 4, byte 1
zero.

Word 5
the first bit represents the value true when 1, and false when 0. The remaining bits of the word are not used.

The format of a subscripted global character item is shown in Figure 43.

Key ₍₁₎		
Key ₍₂₎		
Type Code (14)	Location of Next Hash Synonym	
Max SS Value	Status Flag	
Location of First SS Trailer		

Figure 43. Subscripted global character item

Words 1 and 2
8-character alphanumeric key.

Word 3, byte 1
type code - hexadecimal 14.

Word 3, bytes 2-4
location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 2.

Word 4, byte 1
maximum allowable subscript value defined for this item (1-255).

Word 4, byte 2

bit 0
redefinition flag. If 0, the symbol has not been reprocessed by Phase IIA. In Phase IIA a zero in this bit causes the value of the item to be reset to null (null character string) and the bit to be set on. The item is set to null by zeroing the link to the first subscript trailer or the length field, as the case may be.

bits 1-7
not used.

Word 4, bytes 3 and 4
not used, for ease of processing GBL and LCL symbols together.

Word 5
location of the first subscript trailer, expressed as a 24-bit increment to the base of working segment 2.

The format of the subscript trailer is shown in Figure 44.

This SS Value	Location of Next SS Trailer
Length of Character Value	
Value (1)	
Value (2)	

Figure 44. Trailer item for subscripted global trailer item

Word 1, byte 1
value of the subscript represented by this trailer. Trailers are present only for those subscripts which are not null.

Word 1, bytes 2-4
location of the next subscript trailer, expressed as a 24-bit increment to the base of working segment 2.

Word 2
length of the character string value, 0 to 8.

Words 3 and 4
character string value (up to 8 bytes).

The format of an unsubscripted global character item is shown in Figure 45.

Key ₍₁₎		
Key ₍₂₎		
Type Code (14)	Location of Next Hash Synonym	
0	Status Flag	
Length of Character Value		
Value (1)		
Value (2)		

Figure 45. Unsubscripted global character item

The words have the same meaning as when the item is subscripted, except:

Word 4, byte 1
zero.

Word 5
length of the character string value,
0 to 8.

Words 6 and 7
character string value (up to 8
bytes).

Sequence Symbol Item

Sequence symbols are used primarily within macro definitions and are therefore described in the macro level dictionary. However, they may be used outside of macros, in which case the item for them are placed in the main dictionary in the same format as used in the macro level dictionary.

LOGICAL ORDER FILE (LOF)

The LOF is a series of variable length entries linked to represent the true generation path of the assembly with all conditionalities evaluated and expanded; i.e., it represents the logical order of the assembly. In addition, it contains entries which represent diagnostic error messages. Each entry contains an identifying directive code. The description of the operation code table contains a list of these codes.

The LOF is dynamically constructed in segment 2 during Phase I by the statement analyzer module (CEVST). During Phase IIA, the statement analyzer module supplements the file with insertions generated by macro expansions. During Phase IIB, the LOF chain is further supplemented by alignment entries to account for object program space unused because of the boundary alignment demanded by machine instructions and constants.

The LOF contains an entry for the following types of items. The format of each entry is shown separately following this list.

- Machine Operations
- Macro Instructions
- Literal Origins
- DC/DS/CXD/DXD Definitions
- ORG Statements
- USING Statements
- SET Statements
- PRINT Statements
- Alignment Specifications
- Diagnostic Messages
- MNOTE* Statements

TITLE Statement
END Statement
General format

Machine Operation Entry

The format of a machine operation entry is shown in Figure 46.

Directive Code		Machine Op Code	Increment to Location Counter
A	B	Location of Next LOF Entry	
Increment to Operand Field		Location of Value Item (if Name Present)	
Location of Source Statement			
Location of Literal Operand			

Figure 46. Machine operation entry

Word 1, byte 1
directive code - hexadecimal 00 through 0B; indicates format of machine instruction. The machine instruction directive codes are listed in the description of the operation code table.

Word 1, byte 2
hexadecimal machine operation code

Word 1, bytes 3 and 4
increment to the location counter. This is the amount of storage required by this machine instruction, expressed in bits.

Word 2, byte 1
status indicator fields:

bit 0
if on, indicates that the source statement contained a symbol in the name field.

bit 1
if on, indicates that the operand field contains a literal. The location of the first character of the literal is attached to the entry in word 5.

bit 2
not used for machine instructions.

bit 3
indicates that the statement was generated by a macro instruction.

bits 4-7
if the source statement operation field contained an extended mnemonic, the M1 field of the machine instruction corresponding to the extended mnemonic is carried here.

Word 2, bytes 2-4
location of the next LOF entry, expressed as a 24-bit increment to the base of working segment 2.

Word 3, byte 1
increment to be added to the location of the source statement to produce the address of the first character of the operand field. If the statement has no operand, this field will be zero.

Word 3, bytes 2-4
location of the relocatable value item in the main dictionary corresponding to the symbol in the name field, when such a symbol is present (see word 2, byte 1, bit 0).

Word 4
location of the first byte of the symbolic source statement, exclusive of control bytes.

Word 5
location of the first byte of the literal operand (excluding the =). This word is present only when bit 1 of word 2, byte 1 is on.

Macro Instruction Entry

The format for macro instruction entry is shown in Figure 47.

Directive Code (0C)		
A	B	Location of Next LOF Entry
Increment to Operand Field		
Location of Source Statement		
Location of Macro-Name Item in Dictionary		

Figure 47. Macro instruction entry

Word 1, byte 1
directive code - hexadecimal 0C.

Word 1, bytes 2-4
not used for macro instructions.

Word 2, byte 1

bits 0-2 and 4-7
not used.

bit 3
indicates that the statement was generated by a macro instruction.

Word 2, bytes 2-4
location of the next LOF entry, expressed as a 24-bit increment to the base of the working segment 2.

Word 3, byte 1
increment to be added to the location of the source statement to produce the address of the first character of the operand field.

Word 3, bytes 2-4
not used for macro instructions.

Word 4
location of the first byte of the symbolic source statement, exclusive of control bytes.

Word 5
location of the macro name item in the macro name dictionary.

Literal Origin Entry

The format for a literal origin entry is shown in Figure 48.

Directive Code (26)		Location of First Literal in Pool
A	B	Location of Next LOF Entry
		Location of Value Item (if Name Present)
Location of Source Statement		

Figure 48. Literal origin entry

Word 1, byte 1
directive code - hexadecimal 26.

Word 1, bytes 2-4
location of literal trailer item in the permanent dictionary, expressed as a 24-bit increment to the base of working segment 2. The location is that of the first trailer item for this LTRG number, and represents the first literal in the pool for this LTRG.

Word 2, byte 1
status indicator fields:

bit 0
if on, indicates that the LTOrg statement contained a symbol in the name field.

bit 1
not used for LTOrg.

bit 2
always on for LTOrg.

bit 3
indicates that the statement was generated by a macro instruction or by the assembler for control purposes.

bits 4-7
not used for LTOrg.

Word 2, bytes 2-4
location of the next LOF entry, expressed as a 24-bit increment to the base of working segment 2.

Word 3, byte 1
not used for LTOrg.

Word 3, bytes 2-4
location of the relocatable value item in the main dictionary corresponding to the symbol in the name field, when such a symbol is present.

Word 4
location of the first byte of the symbolic source statement, excluding control bytes.

Constant-Definition Entry

The format for a constant-definition entry is shown in Figure 49.

Directive Code		Location of Constant Item
A	B	Location of Next LOF Entry
Increment to Operand Field		Location of Value Item (if Name Present)
Location of Source Statement		

Figure 49. Constant-definition entry

Word 1, byte 1
directive code - hexadecimal 15 (DC), 17 (DS), 3B (CXD), or 3C (DXD).

Word 1, bytes 2-4
location of the constant item prepared by CSCAN, expressed as a 24-bit increment to the base of working segment 2. If the modifiers of the constant operand were unevaluable in Phases I or IIA, evaluation and preparation of the constant item are deferred until Phase IIB. In such a case, this field is zero until Phase IIB, and bit 2 of word 2, byte 1 is set.

Word 2, byte 1
status indicator fields:

bit 0
if on, indicates that the source statement contained a symbol in the name field.

bit 1
not used for DC/DS/CXD/DXD.

bit 2
if on, indicates evaluation of the constant is required. If word 1, bytes 2-4 is zero, the constant must be evaluated in its entirety. If a constant item is present, however, this bit indicates that the duplication factor was zero and that location counter alignment must be performed.

bit 3
indicates that the statement was generated by a macro instruction.

bits 4-7
not used by DC/DS/CXD/DXD.

Word 2, bytes 2-4
location of the next LOF entry, expressed as a 24-bit increment to the base of working segment 2.

Word 3, byte 1
increment to be added to the location of the source statement to produce the address of the first character of the operand field.

Word 3, bytes 2-4
location of the relocatable value item in the dictionary corresponding to the symbol in the name field, when such a symbol is present, expressed as a 24-bit increment to the base of working segment 2.

Word 4
location of the first byte of the source statement, excluding control bytes. Zero for second and successive operands of a multiple operand constant.

Origin Entry

The format for origin entry is shown in Figure 50.

Directive Code (27)		
A	B	Location of Next LOF Entry
Increment to Operand Field		
Location of Source Statement		
Control Sect No.	Location Counter Value	

Figure 50. Origin entry

Word 1, byte 1
directive code - hexadecimal 27.

Word 1, bytes 2-4
not used for ORG statements.

Word 2, byte 1
status indicator fields.

bits 0-1 and 4-7
not used.

bit 2
always on for ORG statements, indicating a discontinuity in the assignment of location counter values.

bit 3
indicates that the source statement was generated by a macro instruction, if on.

Word 2, bytes 2-4
location of next LOF entry, expressed as a 24-bit increment to the base of working segment 2.

Word 3, byte 1
increment to be added to the location of the source statement to produce the address of the first character of the operand field.

Word 3, bytes 2-4
not used for ORG statements.

Word 4
location of the first byte of the source statement, excluding control bytes.

Word 5

control section number and location counter displacement representing the new origin specified by the ORG statement.

USING Entry

The format for USING entry is shown in Figure 51.

Directive Code (36)		
A	B	Location of Next LOF Entry
Increment to Operand Field		
Location of Source Statement		
Location Counter Value; Register Base Value		

Figure 51. USING entry

Word 1, byte 1
directive code - hexadecimal 36.

Word 1, bytes 2-4
not used for USING statements.

Word 2, byte 1
status indicator fields:

bits 0-2
not used.

bit 3
if on, indicates that the statement was generated by a macro instruction.

bits 4-7
type code for base value. Controls contents of word 5:

- 0 - absolute value
- 1 - relocatable value
- 3 - external value

word 2, bytes 2-4
location of the next LOF entry, expressed as a 24-bit increment to the base of working segment 2.

Word 3, byte 1
increment to be added to the location of the source statement to produce the address of the first character of the operand field.

Word 3, bytes 2-4
not used for USING statements.

Word 4
location of the first byte of the source statement, excluding control bytes.

Word 5
set to location counter value in effect when USING statement is encountered by Phase IIB. Later, set to value of register base specified in the USING statement by Phase IIC. If absolute, it is carried as a 32-bit signed integer. If relocatable, it is carried as an 8-bit section number and a 24-bit displacement. If external, it is zero. This word is used to assist in the formatting of the listed USING statement only.

PRINT Entry

The format for PRINT entry is shown in Figure 52.

Directive Code (2C)	Status Indicators	
	Location of Next LOF Entry	
Increment to Operand Field		
Location of Source Statement		

Figure 52. PRINT entry

Word 1, byte 1
directive code - hexadecimal 2C.

Word 1, byte 2
print status indicators. Bit is set if operand is specified. Status of operands not specified is carried from the previous PRINT statement.

- bit 0 - not used
- bit 1 - FULLGEN
- bit 2 - ON
- bit 3 - OFF
- bit 4 - GEN
- bit 5 - NOGEN
- bit 6 - DATA
- bit 7 - NODATA

Word 1, bytes 3-4
not used.

Word 2, byte 1
status indicator fields:

bits 0-2 and 4-7
not used.

bit 3
if on, indicates statement generated by a macro.

Word 2, bytes 2-4
location of the next LOF entry, expressed as a 24-bit increment to the base of working segment 2.

Word 3, byte 1
increment to be added to the location of the source statement to produce the address of the first character of the operand field.

Word 3, bytes 2-4
not used by PRINT.

Word 4
location of the first byte of the source statement, excluding control bytes.

SET Entry

The format for SET entry is shown in Figure 53.

Word 1, byte 1
directive code - hexadecimal 30, 31, or 32 for SETA, SETB, or SETC respectively.

Word 1, byte 2
subscript value, if any, of global variable.

Word 1, bytes 3-4
not used, zero.

Directive Code		SS Value	
A	B	Location of Next LOF Entry	
Increment to Operand Field		Location of Global Item in Dictionary	
Location of Source Statement			
Value (1)			
Value (2)			

Figure 53. SET entry

Word 2, byte 1
half-byte A - status indicator fields:

bit 0
always off; even though symbols appear in the name field of SET statements, the symbols are not subject to the assignment of location counter values.

bit 1-2
not used.

bit 3
if on, indicates that the statement was generated by a macro instruction.

half-byte B

bits 4-7
for GBLC symbols only, the length of the character string value (0 to 8 bytes).

Word 2, bytes 2-4
location of the next LOF entry, expressed as a 24-bit increment to the base of working segment 2.

Word 3, byte 1
increment to be added to the location of the source statement to produce the address of the first character of the operand field.

Word 3, bytes 2-4
if the SET involves a global symbol, this field contains the location of the global variable item in the permanent dictionary, expressed as a 24-bit increment to the base of working segment 2. Phase IIA reinstates the values of global symbols in response to entries in the GSM chain. This field, the subscript, and the value are used to accomplish the processing.

Word 4
location of the first byte of the source statement, excluding control bytes.

Word 5
for SETA, a 32-bit signed value. For SETB, a one-bit value in bit 0. For SETC, the first four bytes of the character string.

Word 6
for SETC only, the last four bytes of the character string. The value words are also used to assist in formatting SET lines on the printed listing.

Alignment Specification Entry

The format of an alignment specification entry is shown in Figure 54.

In order to keep the location counter computations exactly in step during Phase III with the values previously computed in Phase IIB, this item is inserted in the logical order file during Phase IIB. Phase III will advance the location counter by the amount specified, and, if directed, will produce generated zero-fill for the skipped bits. Phase IIB inserts this item whenever an instruction, DC, DS, CNOP, or CCW, requires adjustment of the location counter to a given boundary.

Directive Code (37)		Zero Fill Flag	Number of Bits
A	B	Location of Next LOF Entry	

Figure 54. Alignment specification entry

Word 1, byte 1
directive code - hexadecimal 37.

Word 1, byte 2
if nonzero, indicates to Phase III that the bits skipped over during alignment are to be set to zero. This is the case when the statement following alignment is one that generates bits. If zero, the bits skipped over are not processed.

Word 1, byte 3-4
number of bits to be added to the location counter to accomplish the desired alignment.

Word 2, byte 1
half-byte A - status indicator fields:

bit 0-2
not used.

bit 3
always on; indicates the statement was generated by a macro instruction.

half-byte B
not used.

Word 2, byte 2-4
location of the next LOF entry, expressed as a 24-bit increment to the base of working segment 2.

Diagnostic Message Entry

The format of a diagnostic message entry is shown in Figure 55.

The listing routine in Phase III can format a print line for a diagnostic message directly from the logical order file item, using the text location table to get the length and severity of the message.

When in nonconversational mode, the diagnostic message processor module (CEVDX) merely records the diagnostic for later inclusion in the assembly listing, as opposed to issuing the message immediately through the command language interpreter facilities. This is facilitated by the insertion of a diagnostic message entry into the logical order file preceding the entry for the source line to which the comment applies during Phases I and IIA.

Directive Code=38	Length of Source Var or MNOTE Msg	Variable Info No. or Severity Code	Message No. (0 if MNOTE)
	Location of Next LOF Entry		
Statement Number			
Location of Source Text for Variable Information or MNOTE Operand*			

* Present When Variable Sources Text is Required for Message (Variable Information No. =0)

Figure 55. Diagnostic message entry

- Word 1, byte 1
directive code - hexadecimal 38.
- Word 1, byte 2
when required, length of source text to be inserted in the message (0-8 bytes), or length of an MNOTE operand.
- Word 1, byte 3
when required, index number of the standard item of variable information to be inserted into the message; severity code, when MNOTE.
- Word 1, byte 4
diagnostic message number. Zero if MNOTE.
- Word 2
location of next LOF entry, expressed as a 24-bit increment to the base of working segment 2.
- Word 3
number of statement in error. Carried as a 7-digit signed packed decimal integer. This field is used for messages which are printed at the end of the listing.
- Word 4
location of variable source text or MNOTE operand (32 bits).

MNOTE* Entry

The format for MNOTE* entry is shown in Figure 56.

Directive Code (3A)		Length of MNOTE Text
	Location of Next LOF Entry	
Increment to Operand Field		
Location of Source Statement		

Figure 56. MNOTE* entry

- Word 1, byte 1
directive code - hexadecimal 3A.
- Word 1, bytes 2-3
not used.
- Word 1, byte 4
length of text in MNOTE* instructions operand.
- Word 2, byte 1
not used.
- Word 2, bytes 2-4
location of the next LOF entry, expressed as a 24-bit increment to the base of working segment 2.
- Word 3, byte 1
increment to be added to the location of the source statement to produce the address of the first character of the operand field.
- Word 3, bytes 2-4
not used.
- Word 4
location of the first byte of the source statement, excluding control bytes.

TITLE Entry

The format for a TITLE entry is shown in Figure 57.

- Word 1, byte 1
directive code - hexadecimal 35.
- Word 1, bytes 2-3
not used.

Word 1, byte 4
length of text which makes up title.

Word 2, byte 1
status indicator field:

bits 0-2 and 4-7
not used.

bit 3
if on, indicates statement was generated.

Word 2, bytes 2-4
location of the next LOF entry, expressed as a 24-bit increment to the base of working segment 2.

Word 3, byte 1
increment to be added to the location of the source statement to produce the address of the first character of the operand field.

Word 3, bytes 2-4
not used.

Word 4
location of the first byte of the source statement, excluding control bytes.

Word 5
location of text specified by character string in TITLE (32 bits).

Directive Code (35)		Length of Title Text
	Location of Next LOF Entry	
Increment to Operand Field		
Location of Source Statement		
Location of Text for Title		

Figure 57. TITLE entry

END Entry

The format for an END entry is shown in Figure 58.

Word 1, byte 1
Directive code - hexadecimal 1A.

Word 1, bytes 2-4
not used for END statements.

Word 2, byte 1
status indicator fields:

bits 0-2 and 4-7
not used.

bit 3
if on, indicates statement was generated.

Word 2, bytes 2-4
location of the next LOF entry, expressed as a 24-bit increment to the base of working segment 2.

Word 3, byte 1
increment to be added to the location of the source statement to produce the address of the first character of the operand field.

Word 3, bytes 2-4
not used for END statements.

Word 4
location of the first byte of the source statement, excluding control bytes.

Word 5
location counter value associated with the module entry point.

Directive Code (1A)	
	Location of Next LOF Entry
Increment to Operand Field	
Location of Source Statement	
Control Section No.	Location Counter Value

Figure 58. END entry

General Format for LOF Entry

The format for the LOF entry is shown in Figure 59.

Note: This LOF entry format is used by all instructions not previously described.

Word 1, byte 1
directive code.

Word 1, bytes 2-4
not used.

Word 2, byte 1
status indicators:

bit 0
if on, indicates the source statement contained a symbol in the name field.

bit 1

if on, indicates operand field contained a literal whose address is in word 5.

bit 2

if on, indicates further processing in Phase IIB is required.

bit 3

if on, indicates statement was generated either by a macro or the assembler.

bits 4-7

not used.

Word 2, bytes 2-4

location of the next LOF entry, expressed as a 24-bit increment to the base of working segment 2.

Word 3, byte 1

increment to be added to the location of the source statement to produce the address of the first character of the operand field.

Word 3, bytes 2-4

location of the dictionary item corresponding to the symbol in the name field, when such a symbol is present, expressed as a 24-bit increment to the base of working segment 2.

Word 4

location of the first byte of the source statement, excluding control bytes.

Word 5

location of literal if one is present in operand of the source statement; this word not present if no literal in the statement.

Directive Code		
A	B	Location of Next LOF Entry
Increment to Operand Field		Location of Dictionary Item (If Name Present)
Location of Source Statement		
Location of Literal When One is Present		

Figure 59. General format for LOF entry

GLOBAL SECTION MACRO CHAIN (GSM)

The GSM chain is created during Phase I. The chain points to control section statements (START, CSECT, DSECT, PSECT, COM) and the following, in order of occurrence:

- Macro instructions.
- Global SET statements.
- ENTRY, PRINT, LTORG, USING, DROP statements.

During Phase IIA, the GSM chain is followed, ignoring LTORG, USING, and DROP statements, to perform one of the following types of processing:

- Posting a new current control section.
- Updating a global SET symbol dictionary definition.
- Expanding a macro instruction.
- Resetting the PRINT status.

During this processing, macro instructions and GBL and SET entries are removed from the GSM chain; control section declaratives and ENTRY, PRINT, LTORG, USING, and DROP statements generated by macro expansions are inserted into the chain.

During Phase IIB, entries are added to the GSM chain for generated control section statements and LTOrgs.

During Phase IIC, the GSM chain is passed over in order. ENTRY trailer items are formed from ENTRY statements; USING and DROP statements are used to adjust the base register table. PRINT statements are used to amend the listing control specifications. LTOrg statements are used to update the LTOrg number. The entries for all of these except USING and DROP statements are then removed from the GSM chain. USING and DROP GSMs are replaced with base register locator items. Wherever a control section statement is encountered, the base register table is put out to provide initial conditions for the control section.

During Phase III, the GSM chain is used as a guide to permit processing in control section order.

The format of a GSM entry is shown in Figure 60.

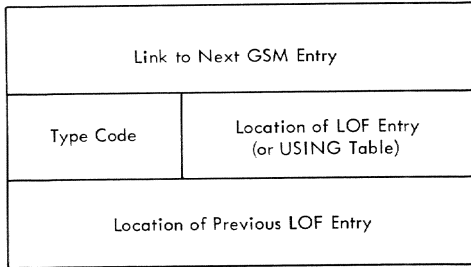


Figure 60. GSM entry format

Word 1
location of next GSM entry (32 bits).

Word 2, byte 1
type code, assigned as follows:

- 0 - Section statement
- 1 - Macro instruction
- 2 - GBLx statement
- 3 - SETx statement (global symbol)
- 4 - USING
- 5 - DROP
- 6 - Base register table locator
- 7 - ENTRY
- 8 - PRINT
- 9 - LTORG
- A - Generated statement

Word 2, bytes 2-4
pointer. For all types except 6, this is the address of the logical order file entry for the statement to be processed, expressed as a 24-bit increment to the base of working segment 2. For type 6, it is the address of the base register table in effect at the section break. The address is expressed as a 24-bit increment to the base of working segment 1.

Word 3
for all types except 6, this is the 32-bit address of the logical order file entry for the statement preceding that to be processed. This address is required for the insertion of diagnostic message entries into the LOF for the statement being processed.

MACRO NAME DICTIONARY

The macro name dictionary is used to establish and develop symbol definitions used as macro names. It is a variable length array of linked items dynamically constructed in Segment 2 by the macro definition processor module (MACDEF) during Phase I. It is used by the collect and identify operation code module (GETOP) and the macro reference processor module (MACREF).

Symbols in the macro name dictionary are located by applying a hashing algorithm to the alphameric value of the symbol. The result is an index value to a table of addresses, of which each address space contains the address of the most recently constructed dictionary item whose key hashes to this value.

The format of a macro name dictionary entry is shown in Figure 61.

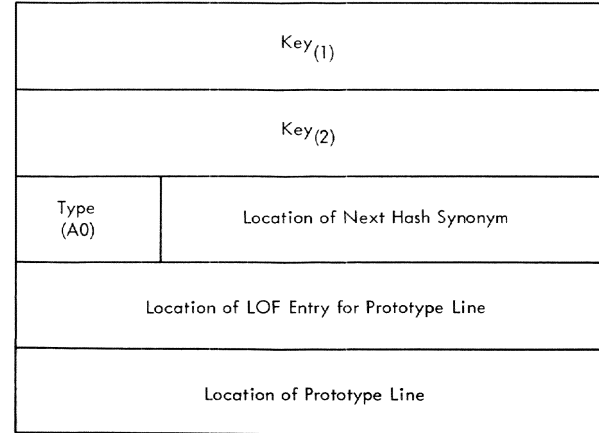


Figure 61. Macro name dictionary item

Words 1 and 2
8-character alphameric key.

Word 3, byte 1
type code - hexadecimal A0.

Word 3, bytes 2-4
location of next item whose key hashes to the same value, carried as a 24-bit increment to the base of working segment 2.

Word 4
location of LOF entry for the macro prototype statement. If zero, the macro definition is not yet in storage.

Word 5
location of the first byte of the macro prototype statement.

OPERATION CODE TABLE

The operation code table is an ordered list of the mnemonic codes used for machine operations and assembler instructions. It is maintained in the common constant area of assembler storage.

Each machine and assembler instruction is represented by an entry in the operation code table. Entries are packed densely and are in alphabetical order.

The format of each entry is shown in Figure 62.

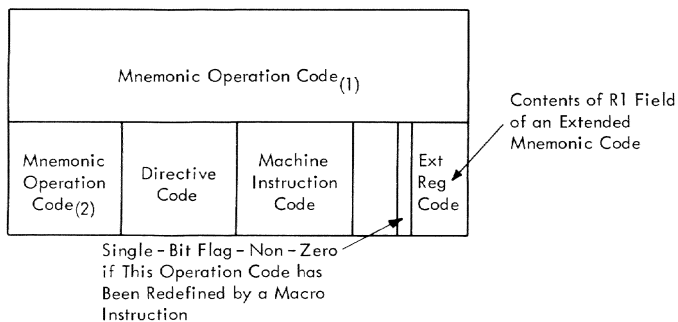


Figure 62. Item format for operation code table entry

- Word 1**
first four characters of mnemonic operation code.
- Word 2, byte 1**
fifth character (if any) of mnemonic operation code. The operation code is left justified in a blank field.
- Word 2, byte 2**
directive code (see Table 10).
- Word 2, byte 3**
hexadecimal machine instruction code, where applicable. Otherwise, zero. (See Table 11.)
- Word 2, byte 4**
- bits 0-2
not used.
 - bit 3
if 1, machine instruction code has been superseded by a macro definition. Mnemonic must then be located in the macro name dictionary. Otherwise, zero.
 - bits 4-7
M1 field value for extended mnemonic operation codes. Zero, if not applicable.

MACHINE OPERATIONS REQUIREMENTS TABLE

The machine operations requirements table is used to check the register and address specifications of certain machine operations for validity.

Table 10. Directive code assignments

Directive Code	Instruction	Directive Code	Instruction
00-0B	Machine Instructions (See Table 11)	25	LCLC
0C	Macro Instruction	26	LTORG
0D	AGO	27	ORG
0E	AIF	28	MACRO
0F	ANOP	29	MEND
10	CCW	2A	MEXIT
11	CNOP	2B	MNOTE
12	COM	2C	PRINT
13	COPY	2D	PSECT
14	CSECT	2E	PUNCH
15	DC	2F	REPRO
16	DROP	30	SETA
17	DS	31	SETB
18	DSECT	32	SETC
19	EJECT	33	SPACE
1A	END	34	START
1B	ENTRY	35	TITLE
1C	EQU	36	USING
1D	EXTRN	37	Alignment Entry
1E	GBLA	38	Diagnostic Message Entry
1F	GBLB	39	Macro Prototype Line
20	GBLC	3A	MNOTE*
21	ICTL	3B	CXD
22	ISEQ	3C	DXD
23	LCLA	3D-3E	Unassigned
24	LCLB	3F	Commentary

Table 11. Machine instruction directive codes

Code	Instruction Type	Assembler Operand Field Format
0	RR	R1, R2
1	RR Extended	R2
2	RR	R1
3	RR	I
4	RX	R1, D2 (X2, B2) R1, S2 (X2)
5	RX Extended	D2 (X2, B2) S2 (X2)
6	RS	R1, R3, D2 (B2) R1, R3, S2
7	RS	R1, D2 (B2) R1, S2
8	SI	D1 (B1), I2 S1, I2
9	SI	D1 (B1) S1
A	SS	D1 (L1, B1), D2 (L2, B2) S1 (L1), S2 (L2)
B	SS	D1 (L, B1), D2 (B2) S1 (L), S2

Certain instructions require specification of an even register number, or a floating point register; others require the operand address to be aligned on a word, halfword, or doubleword boundary. Store type operations must be checked for illegal literal operands. The machine operations requirements table summarizes these specifications for each machine operation code.

The machine operations requirements table is 256 bytes long. An entry is found by indexing into the table with the hexadecimal machine instruction code.

The format of each entry byte in the table is shown in Figure 63.

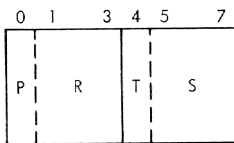


Figure 63. Entry byte format

Bit P (bit 0)

if on, indicates that the address portion of a 4-byte instruction is to be printed in the ADDR 2 field of the listing; if off, in the ADDR 1 field.

field R (bits 1-3)

is the register requirement:

- 0 - No special requirement
- 1 - Even numbered register
- 2 - Floating point register
- 3 - Floating point register 0 or 4
- 4 - Exceptional requirements:

- MP - L2 must be less than L1 and not larger than 7.
- DP - L2 must be less than L1 and not larger than 7.
- SLT - L2 must be 0, 1, 2, 3 or 4; B2 should not be 0, 1, 2, 3 or 4.

bit T (bit 4)

if on, indicates a store type operation.

field S (bits 5-7)

is the storage address requirement:

- 0 - No special requirement
- 1 - Halfword alignment
- 2 - Fullword alignment
- 3 - Doubleword alignment

USING-REGISTER TABLES

Using-register tables are dynamically constructed during Phase IIC in working segment 1 for each section break, USING, and DROP statement. They are used during

Phase III to assign base register and displacement values to machine instructions and S-type address constants.

Each table contains 31 words, an initial word and 15 two-word entries (one entry for each potential base register).

The first word of each table is used initially to communicate the current listing options, the correct LTORG number. When applicable, this word is reused during Phase III to transmit information to the program checkout system (PCS). The location counter value in effect when the table is first used is inserted to inform the PCS of the start of a range of locations over which the listed base registers are effective.

A description of how the table is searched is included under the routine description of USEVAL.

The format of a using-register table is shown in Figure 64.

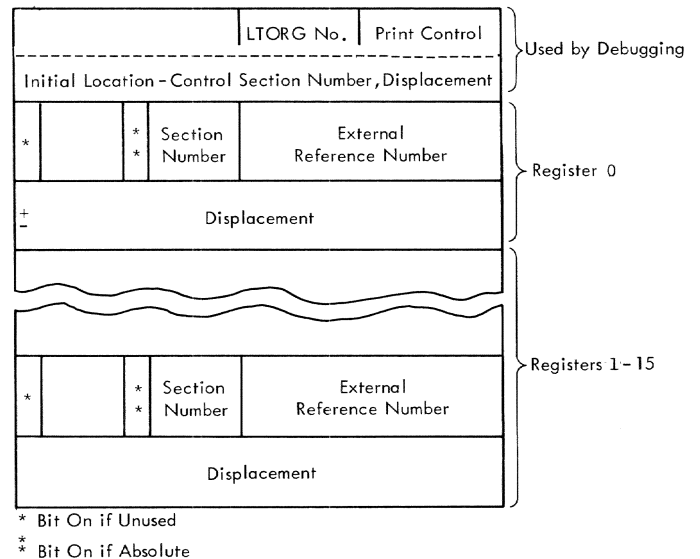


Figure 64. Using-register table format

Word 1

between Phases II and III this word carries a PRINT status indicator, for control of the printed listing in Phase III and a LTORG number. The status indicator defines the print options in effect at the control section break at which the using table becomes effective (ON/OFF, FULLGEN/GEN/NOGEN, DATA/NODATA). The status is carried in byte 4, in the low-order seven bits as follows:

- bit 1 - FULLGEN
- bit 2 - ON
- bit 3 - OFF
- bit 4 - GEN

bit 5 - NOGEN
bit 6 - DATA
bit 7 - NODATA

Each bit is set if the corresponding operand was specified. Options not specified (zero bits) are retained from the previous PRINT status. Byte 3 contains the current LTOrg number. If production of an internal symbol dictionary (ISD) is specified as an assembly option, Phase III replaces the contents of this word with the value of the location counter for the first address at which the using register table becomes effective. This location is expressed as an 8-bit section number and a 24-bit displacement.

Word 1, bytes 1-2
not used.

Word 1, byte 3
LTOrg number in effect at the time of construction of the table.

Word 1, byte 4
PRINT status indicator, for control of the printed listing in Phase III.

Word 2, byte 1
status indicators for register 1:

bit 0
if on, indicates the register is not used as a base register.

bit 7
if on, indicates that the register contains an absolute value as a base, and that the rest of the word is zero.

Word 2, byte 2
if the register contains a simply relocatable base value, the number of the section governing the relocation is entered here. If this is the case, the other fields of this word are zero.

Word 2, bytes 3-4
if the register contains an external value as a base, the external reference number of the pertinent symbol is entered here, and the other fields of the word are zero.

Word 3
if the base value is absolute, the 32-bit signed value is carried here. If the base is simply relocatable, the displacement of the location counter relative to the base of the control section (see word 2, byte 2) is carried here. If the base is external, any absolute portion of the USING expression is carried here (e.g.,

ALPHA+5, where ALPHA is external and +5 is the absolute portion).

Words 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, and 32
these words duplicate the contents of word 2 for base registers 2 through 15, respectively.

Words 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, and 33
these words duplicate the content of word 3 for base registers 2 through 15, respectively.

MACRO LEVEL DICTIONARY (TEMPORARY DICTIONARY)

Temporary macro level dictionaries are used to establish and develop the definitions for variable symbols which occur at a given level of macro expansion. The dictionary for a given macro level is constructed by PARAMAC during Phase IIA when STAN encounters a macro instruction reference.

A temporary macro level dictionary is a variable length array of items dynamically constructed by the assembler in working storage segment 1 during Phase IIA. Its general layout is shown in Figure 65.

The main dictionary contains descriptions of the general characteristics of dictionary items. During Phase IIA the macro level dictionaries occupy contiguous storage in working segment 1. A dictionary is constructed by acquiring the next open locations in this area. Control of available space is maintained by the address pointer AWORK1, which is updated by each routine which requires working storage during macro expansion. At the conclusion of the expansion of a given macro, the space occupied by its dictionary is reclaimed by resetting the value of AWORK1.

Thus, the total amount of storage required for macro dictionaries expands and contracts to the degree that macro instructions are nested. A level counter MLVL indicates the current depth of nesting at all times during Phases I and IIA. If the value of MLVL is zero, the current statement is outside of macro expansions (user code), and no macro dictionary exists.

At each level the dictionary exhibits the same general organization. The first word is reserved for an address pointer to the first word of the dictionary for the preceding (lower) macro level. These pointers are used when the space for a dictionary is recovered at the end of an expansion. A constant pointer, ACTSD, indicates the location of the dictionary currently in use.

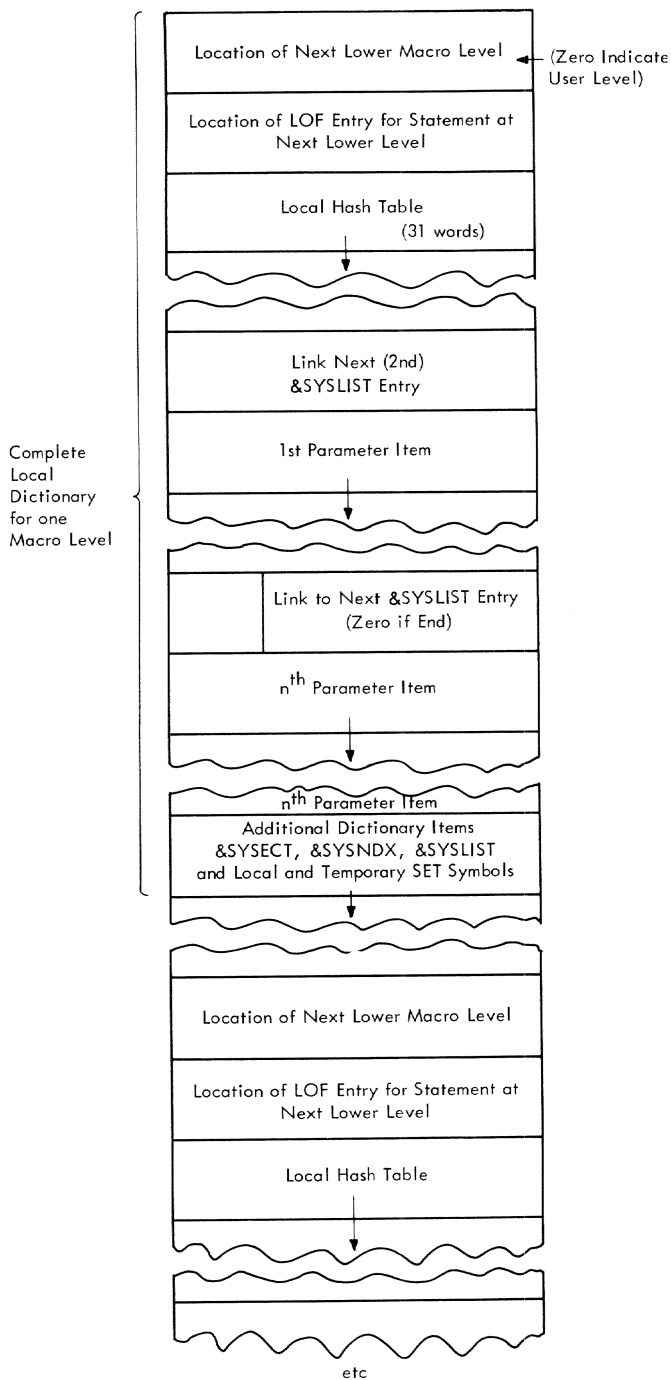


Figure 65. Layout of macro level dictionary

The second word of a macro level dictionary contains a 32-bit address pointer to the LOF entry for the statement at which processing was suspended at the next lower level. This address is used to resume processing at the next lower level when a macro level dictionary is erased.

The next 31 words of a macro level dictionary are reserved for a small hash table. This table functions in the same way as the hash table for the permanent dictionary. It permits the conversion of an 8-character alphameric symbol (key) into the address of an item in the dictionary. Since the incidence of variable symbols is generally low, a smaller table can function effectively for the macro level dictionaries.

Following the hash table, the remainder of a macro level dictionary is filled with items for the variable symbols pertaining to that level. Certain symbols are always present. These are the system variables &SYSLIST, &SYSNDX, &SYSECT, &SYSPSCT, and &SYSSTYP.

&SYSLIST represents an alternate method of specifying the positional parameters of a macro. The &SYSLIST item consists of a basic entry and one trailer for each positional parameter. The storage assignments are such that each &SYSLIST trailer immediately precedes the positional parameter item associated with it. Except for this case, the storage assignment for a given item is a function of the occurrence of the symbol in the macro prototype or model statements and has no special significance.

Item Types

For convenience, descriptive summaries of the items which may appear in a macro level dictionary are given below. More detailed information for each item follows these summaries.

&SYSLIST Item: The purpose of this item is to point to the beginning of the current &SYSLIST chain. The first byte of the fourth word contains the number of &SYSLIST entries (i.e., N*&SYSLIST) and the remainder of the word is the pointer.

&SYSNDX Item: This is a 4-word item, the last word of which contains the macro index value for the current macro level. The value is contained in the item in binary and is derived simply by incrementing a counter by one whenever a macro instruction is encountered.

&SYSECT Item: This also is a 4-word item with the fourth word pointing to the control section item that is the current control section (CCS) when the macro instruction occurs.

&SYSPSCT Item: This item is identical to the &SYSECT item in format; it even uses the same type code. However, the alphameric key is different (&SYSPSCT) and the pointer in the fourth word indicates the control section item in the permanent dic-

tionary which defines the first prototype section in the program. If no prototype section exists, the pointer is zero.

&SYSSTYP Item: This item is a 4-word item with the fourth word pointing to the same control section item as the &SYSECT item. It is used by the string substitution routine (SSCAN) to derive a character string which is a mnemonic for the control section type.

Parameter Item: A parameter item is constructed in the current macro level dictionary by the PARAMAC routine of Phase IIA. A parameter item is made for each symbolic parameter given on the prototype line. The argument string field of the item consists of the character string supplied on the macro instruction line (or by default) in fulfillment of the parameter.

The item contains a 1-word descriptor for each subargument, if any. The descriptor indicates the length in bytes of the subargument, its location from the beginning of the argument character string, and whether the subargument is a selfdefining term.

Sequence Symbol Item: Sequence symbol items are constructed by the statement analyzer in the current local dictionary when the macro level is greater than zero. The item points to the source line on which the sequence symbol appears. Sequence symbol items are constructed in the permanent dictionary when the macro level is zero.

Local Variable Symbol Items: Local variable symbols are those defined by the instructions LCLA, LCLB, and LCLC. The values of such symbols are dependent upon macro level; their dictionary items are accordingly maintained in the current macro level dictionary whenever the macro level is greater than zero, or in the permanent dictionary if they are defined when the macro level is zero. Local variable symbols may be subscripted or unsubscripted. If subscripted, each item contains the maximum permissible subscript and trailers for each non-null value, together with an indication of their respective subscripts. If unsubscripted, each item provides space for the setting of a non-null value. The initial definition of a local symbol creates "null" values. Other values are set by the instructions SETA, SETB, or SETC once the symbol has been defined. At the conclusion of a macro expansion, the values of any local variable symbols in the dictionary for that macro level are lost.

Global Variable Symbol Pointer Items: Global variable symbols are those defined by the instructions GBLA, GBLB, and GBLC. The values of such symbols are independent of macro level; their dictionary items are accordingly maintained in the permanent

dictionary. If a global symbol is defined when the macro level is greater than zero, a special item is entered in the macro level dictionary which points to the corresponding definition of the global symbol in the permanent dictionary.

&SYSLIST Item: (Figure 66)

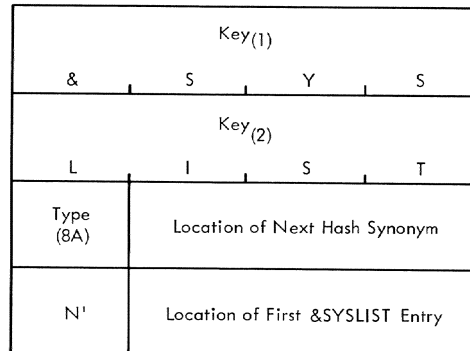


Figure 66. &SYSLIST item

Words 1 and 2

the alphameric characters &SYSLIST.

Word 3, byte 1

type code - hexadecimal 8A.

Word 3, bytes 2-4

location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 1.

Word 4, byte 1

total number of positional parameter items for this macro level. This is the number attribute (N') of &SYSLIST.

Word 4, bytes 2-4

location of the first trailer item [&SYSLIST (1)], expressed as a 24-bit increment to the base of working segment 1.

&SYSNDX Item: (Figure 67)

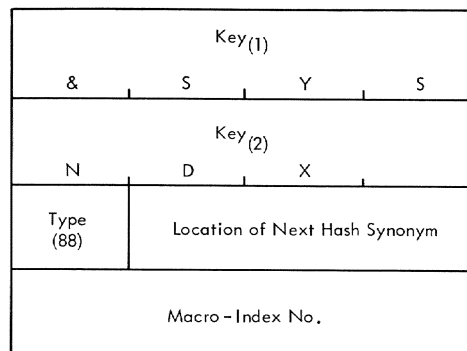


Figure 67. &SYSNDX item

Words 1 and 2
the alphanumeric characters &SYSNDX.

Word 3, byte 1
type code - hexadecimal 88.

Word 3, bytes 2-4
location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 1.

Word 4
number of this macro instruction expansion. This number is set to 1 by Phase IIA initialization, and is incremented by one each time PARAMAC establishes a new macro level dictionary.

&SYSECT Item: (Figure 68)

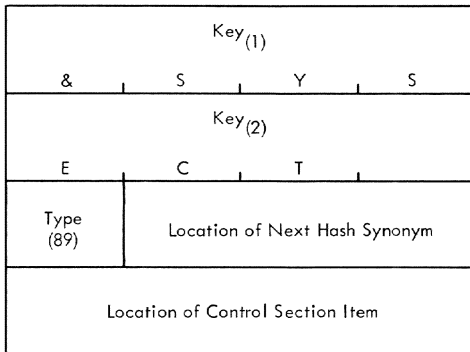


Figure 68. &SYSECT item

Words 1 and 2
the alphanumeric characters &SYSECT.

Word 3, byte 1
type code - hexadecimal 89.

Word 3, bytes 2-4
location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 1.

Word 4
location of the section name item in the permanent dictionary which defines the control section which was current at the time this macro level dictionary was prepared.

&SYSPSCT Item: (Figure 69)

Words 1 and 2
the alphanumeric characters &SYSPSCT.

Word 3, byte 1
type code - hexadecimal 89 (same as &SYSECT item).

Word 3, bytes 2-4
location of next item whose symbol hashes to the same value, expressed as a 24-bit increment to the base of working segment 1.

Word 4
location of the section name item in the permanent dictionary which defines the first PSECT encountered in the source program. Word 4 is zero if no PSECT has been defined.

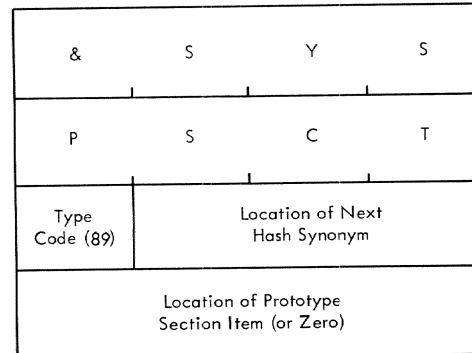


Figure 69. &SYSPSCT item

&SYSSTYP Item: (Figure 70)

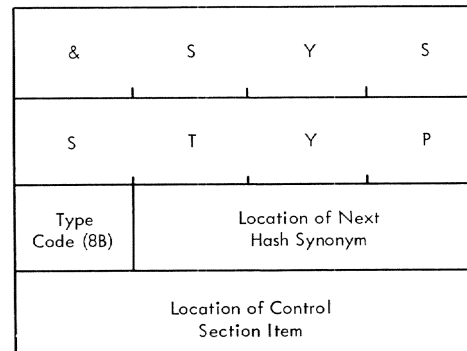


Figure 70. &SYSSTYP item

Word 1 and 2
the alphanumeric characters &SYSSTYP.

Word 3 byte 1
type code - hexadecimal 8B.

Word 3, byte 2-4
location of next item whose symbol hashes to the same value, expressed as a 24-bit increment to the base of working segment 1.

Word 4
location of the section name item in the permanent dictionary which defines the control section which was current at the time this macro level dictionary was prepared.

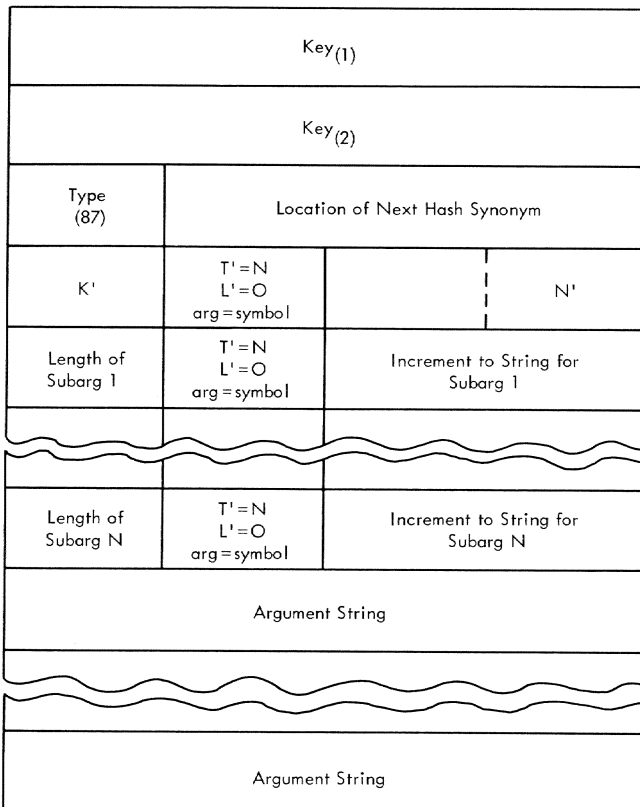


Figure 71. Parameter item (temporary dictionary)

Words 1 and 2

8-character alphameric key.

Word 3, byte 1

type code - hexadecimal 87.

Word 3, bytes 2-4

location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 1.

Word 4

argument control word for the entire parameter.

byte 1

number of characters in the argument string. This is the count attribute (K') of the parameter as a whole.

byte 2

characteristic flags:

bit 0

if 1, this bit indicates that the argument string for the parameter consists entirely of a single self-defining term. This sets the type attribute (T') to the value "N". If 0, the argument is not a self-defining term.

bit 1

if 1, indicates that the length of the argument string is zero, and that the length attribute (L') should be zero. If 0, indicates that the argument string is not null.

bit 2

if 1, indicates that the argument string consists of a single symbol; if 0, indicates that the argument does not meet the syntactic scan requirements for a symbol.

bits 3-7

not used.

byte 3

not used; zero, to permit use of half-word arithmetic on byte 4.

byte 4

number of subarguments (sublists), if any. This count also indicates the number of subargument control words which follow and is the number attribute (N') of the parameter.

Words 5 through 5+N'-1

subargument control words. If any are present, each has the following format:

byte 1

number of characters in the subargument string. This count excludes the terminating comma or parenthesis and is the count attribute (K') of the subargument.

byte 2

characteristic flags. See Word 4, Byte 2.

bytes 3 and 4

increment to be added to the location of the argument string as a whole in order to compute the address of the first character of the subargument string. The argument string address can be computed as $A+16+(4*N')$, where A is the location of the parameter item, and N' is as described in Word 4, Byte 4. Thus, if the argument string were:

(ALPHA,BETA,27+B,=F'4')

the increment to the third subargument would be 12.

Words 5+N' through 5+N'+(K'+3)/4

characters of the argument string. Storage is reserved to the next full word when the character string is not a multiple of four bytes in length.

Sequence Symbol Item: (Figure 72)

Key(1)	
Key(2)	
Type (90)	Location of Next Hash Synonym
Location of Source Statement	

Figure 72. Sequence symbol item

Word 1 and 2

8-character alphanumeric sequence symbol

Word 3, byte 1

type code - hexadecimal 90

Word 3, bytes 2-4

location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 1 if in a macro level dictionary, or working segment 2, if in the permanent dictionary. The distinction is made on the basis of macro level, prior to lookup.

Word 4

location of first byte of the source statement on which the sequence symbol appeared. (This location excludes the control bytes which precede each source statement.)

Arithmetic Item: (Figures 73 and 74)

Key(1)	
Key(2)	
Type (11)	Location of Next Hash Synonym
Max SS Value	Not Used
	Location of Next Subscript

This SS Value	Location of Next Subscript
Value	

} For Each Subscript

Figure 73. Subscripted LCLA item

Key(1)	
Key(2)	
Type (11)	Location of Next Hash Synonym
0	Not Used
Value	

Figure 74. Unscripted LCLA item

Words 1 and 2

8-character alphanumeric key.

Word 3, byte 1

type code - hexadecimal 11.

Word 3, bytes 2-4

location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 1, if in a macro level dictionary, or working segment 2, if in the permanent dictionary. The distinction is made on the basis of macro level prior to lookup.

Word 4, byte 1

maximum allowable subscript value defined for this item (1-255). Zero if item is unscripted.

Word 4, bytes 2-4

not used, for ease of processing GBL and LCL symbols together.

Subscripted Item

Word 5

location of the first subscript trailer, expressed as a 24-bit increment to the base of the applicable working segment.

Unscripted item

Word 5

32-bit signed value of the arithmetic symbol.

Subscript trailers

Word 1, byte 1

values of the subscript represented by this trailer. Trailers are present only for those subscripts which are not null.

Word 1, bytes 2-4
location of next subscript trailer,
expressed as a 24-bit increment to the
base of the applicable working segment.

Word 2
32-bit signed value of the arithmetic
symbol.

Local Boolean Item: (Figures 75 and 76)

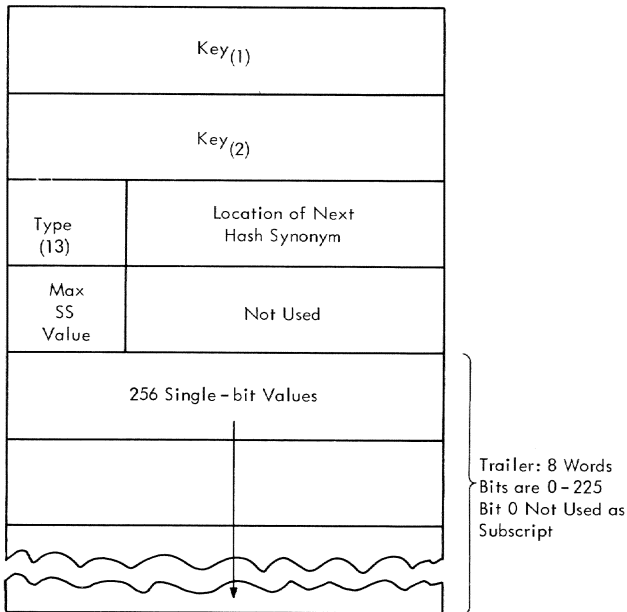


Figure 75. Subscripted LCLB item

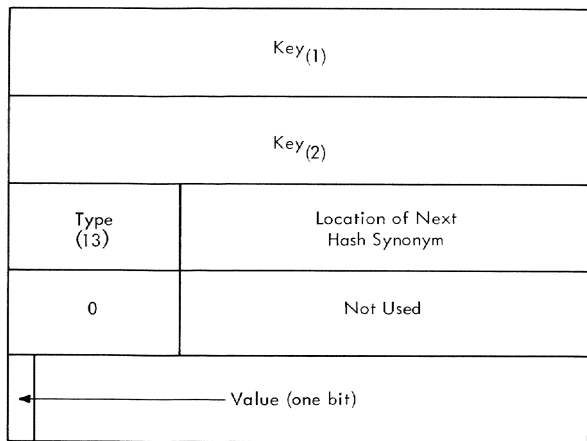


Figure 76. Unsubscripted LCLB item

Words 1 and 2
8-character alphameric key.

Word 3, byte 1
type code - hexadecimal 13.

Word 3, bytes 2-4
location of next item whose key hashes
to the same value, expressed as a 24-
bit increment to the base of working

segment 1, if in a macro level dictionary,
or working segment 2, if in the permanent dictionary. The distinction is made on the basis of macro level, prior to lookup.

Word 4, byte 1
maximum allowable subscript value defined for this item (1-255). Zero if item is unsubscripted.

Word 4, bytes 2-4
unused, for ease of processing GBL and ICL symbols together.

Subscripted item

Words 5-12
256 single-bit values. The first bit represents subscript 0 and is not used. The remainder represent subscripts 1 - 255 and represent true when 1 and false when 0.

Unsubscripted item

Word 5
the first bit represents the value true when 1 and false when 0. The remaining bits of the word are not used.

Local Character Item: (Figures 77 and 78)

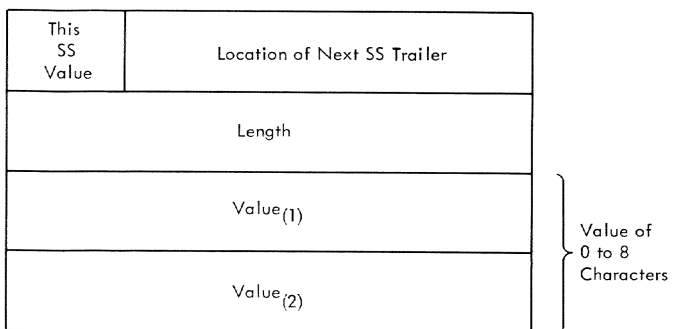
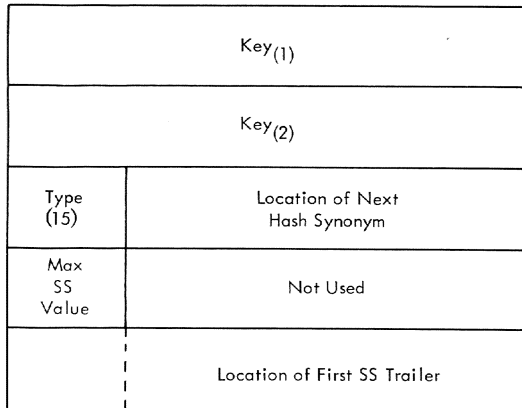


Figure 77. Subscripted LCLC item

Key(1)	
Key(2)	
Type (15)	Location of Next Hash Synonym
0	Not Used
Length	
Value (1)	
Value (2)	

Figure 78. Unsubscripted LCLC item

Words 1 and 2

8-character alphaneric key.

Word 3, byte 1

type code - hexadecimal 15.

Word 3, bytes 2-4

location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 1, if in a macro level dictionary, or working segment 2, if in the permanent dictionary. The distinction is made on the basis of macro level, prior to lookup.

Word 4, byte 1

maximum allowable subscript value defined for this item (1-255). Zero, if item is unsubscripted.

Word 4, bytes 2-4

unused, for ease of processing GBL and LCL symbols together.

Subscripted item

Word 5

location of the first subscript trailer, expressed as a 24-bit increment to the base of the applicable working segment.

Unsubscripted item

Word 5

length of the character string value, 0 to 8 bytes.

Words 6 and 7

character string value (up to 8 bytes).

Subscript trailers

Word 1, byte 1

value of the subscript represented by this trailer. Trailers are present only for those subscripts which are not null.

Word 1, bytes 2-4

location of the next subscript trailer, expressed as a 24-bit increment to the base of the applicable working segment.

Word 2

length of the character string value, 0 to 8 bytes.

Words 3 and 4

character string value (up to 8 bytes).

Global Variable Symbol Pointer Item:
(Figure 79)

Key(1)	
&	
Key(2)	
Type (40)	Location of Next Hash Synonym
Location of GBL Item in Main Dictionary	

Figure 79. GBLA, GBLB or GBLC item in macro level dictionary

Words 1 and 2

8-character alphaneric key.

Word 3, byte 1

type code - hexadecimal 40.

Word 3, bytes 2-4

location of next item whose key hashes to the same value, expressed as a 24-bit increment to the base of working segment 1.

Word 4

location of corresponding global item in permanent dictionary (32 bits).

Comments

Summary of item type codes in macro level dictionaries:

```

00010001    LCLA
00010011    LCLB
00010101    LCLC
01000000    GBL pointer item
10000111    Parameter
10001000    &SYSNDX
10001001    &SYSECT and &SYSPSCT
10001010    &SYSLIST
10001011    &SYSSTYP
10010000    Sequence symbol

```

SOURCE LINE STORAGE CONTROL

Source lines received from external sources are copied into the assembler's own working storage in working segment 3. The lines are of variable length and are preceded by control information relating to the length, sequence number, and sequential order of the line. This control information is used by REED in resolving continuation lines, and in processing library statements.

Continuation lines are appended to a copy of the basic line by REED. The resulting statement is prefixed with the same line number and (concatenated) length information described above. However, the statement is not linked to any others. Access to it is through the logical order file only.

The format of source statement control information is shown in Figure 80.

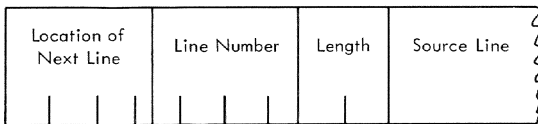


Figure 80. Source statement control information format

Bytes 1-4

location of the next sequential source line. These are the unprocessed lines before continuation lines have been appended. The next sequential line is used in writing the source listing. The address is carried as 32-bits. None of these fields are word-oriented.

Bytes 5-8

line number. The line number is carried as a 7-digit signed packed decimal number; thus:

XXXXXXX+

Bytes 9 and 10

length of the line in bytes.

Bytes 11-n

source line.

PSEUDO-DICTIONARY ITEM FOR CURRENT LOCATION COUNTER

A simulated dictionary item is maintained to fulfill references to the location counter (asterisk notation). This item is not part of the main dictionary since its location is always known and it is not looked up in the normal way (hashing the symbol). However, the existence of this item frequently permits the asterisk notation to be treated as a normal relocatable quantity.

The format of a simulated item for location counter references is shown in Figure 81.

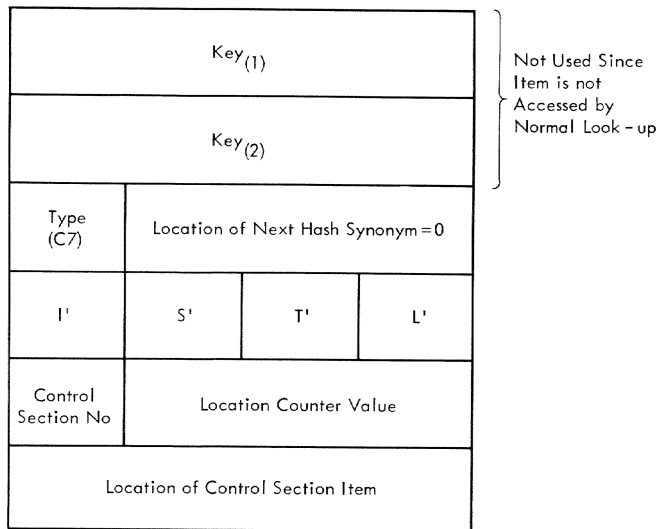


Figure 81. Simulated item for location counter references

Words 1 and 2

not used, since item is not accessed by hash table lookup.

Word 3, byte 1

type code - hexadecimal C7.

Word 3, bytes 2-4

not used, zero; to maintain format of normal dictionary items.

Word 4, bytes 1 and 2

integer and scale attribute fields - not used; zero.

Word 4, byte 3

type attribute - the alphameric character. Not yet determined.

Word 4, byte 4

length attribute. L' is that of the current machine instruction or constant, except in the case of the statement A EQU *, where L' is 1.

Word 5, byte 1
number of the control section which the location counter currently represents.

Word 5, bytes 2-4
current displacement of the location counter within the control section.

Word 6
location of the section name item in the permanent dictionary for the current control section.

Alignment Code Field is filled as follows:

- Bit 0
0 = Alignment Required
1 = No Alignment Required
- Bits 1-2 Specify Alignment:
00 - By Byte
01 - By Halfword
10 - By Word
11 - By Doubleword

CONSTANT ITEM FORMAT

(See Figures 82 and 83.)

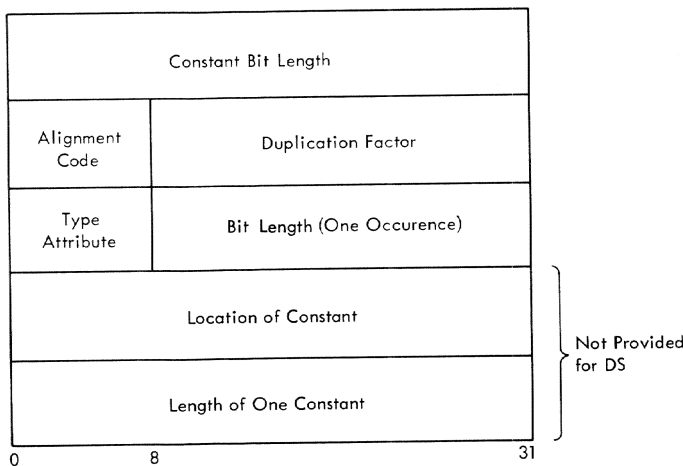


Figure 82. Constant item (address constant)

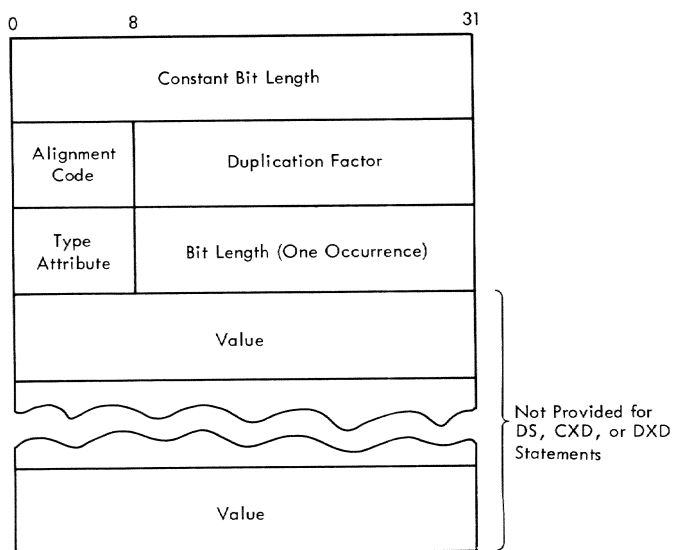


Figure 83. Constant item (other than address constants)

Type Attribute field contains one of the following EBCDIC characters:

- A A-type address constant, implied length, aligned.
- B Binary constant.
- C Character constant.
- D Long floating-point constant, implied length, aligned.
- E Short floating-point constant, implied length, aligned.
- F Fullword fixed-point constant, implied length, aligned.
- G Fixed-point constant, explicit length.
- H Halfword fixed-point constant, implied length, aligned.
- K Floating-point constant, explicit length.
- P Packed decimal constant.
- Q Q-type address constant, implied length, aligned.
- R A-, Q-, R-, S-, V-, or Y-type address constant, explicit length.
- S1 S-type address constant, implied length, aligned.
- V V-type address constant, implied length, aligned.
- X Hexadecimal constant.
- Y Y-type address constant, implied length, aligned.
- Z Zoned decimal constant.
- # R-type address constant, implied length, aligned.

VIRTUAL MEMORY MANAGEMENT TABLE (VMTABLE)

VMTABLE is used by the virtual memory management routines, VMGET, VMFREE, and VMCLEAN, in managing virtual storage for the assembler. By keeping a record of virtual storage obtained as the result of GETMAIN macro instructions issued by VMGET, the number of GETMAINS and FREEMAINS issued and the amount of virtual storage obtained from the system is minimized. The virtual memory management routines keep this record in VMTABLE.

VMTABLE is a contiguous area in PSECT CEVPAR consisting of 64 three-word blocks. Each block is a member of one of four chains within VMTABLE. The chain headers exist outside of VMTABLE in PSECT CEVPAR and point to the first block of their par-

ticular chain within VMTABLE. The names of the chain headers and the corresponding chain functions are:

VMGOTTEN -- Provides a record of areas obtained by the GETMAIN macro instruction and as yet unused by the assembler.

VMASSIGN -- Provides a record of areas obtained by the GETMAIN macro instruction and in use by the assembler.

VMFREED -- Provides a record of areas no longer in use by the assembler and available for return via FREEMAIN.

VMENTRYS -- Provides a pointer to unused VMTABLE blocks.

Formats of VMTABLE Entries

The format and contents of blocks in the four chains in VMTABLE are illustrated in Figures 84, 85, and 86.

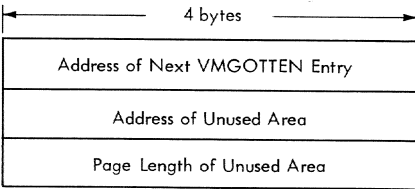


Figure 84. Contents of VMGOTTEN block

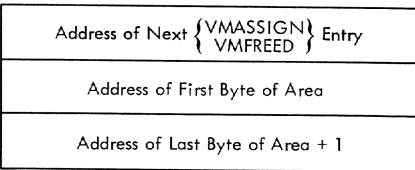


Figure 85. Contents of VMASSIGN and VMFREED blocks

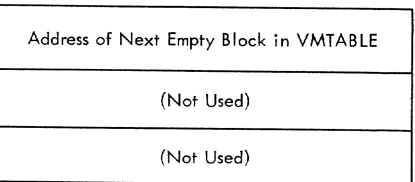


Figure 86. Contents of VMENTRYS block

Both VMGOTTEN and VMASSIGN entries are built by VMGET. VMGET never issues a GETMAIN macro for less than one segment (256 pages) of virtual storage. If invoked to supply less than that, VMGET acquires a complete segment. VMGET then places the address of the assigned pages in a VMASSIGN

entry and the address of the unused pages in a VMGOTTEN entry. If invoked to supply more than one segment, VMGET issues a GETMAIN for the desired amount, then adds a VMASSIGN entry for the storage obtained and assigned. If invoked to supply less than one segment and VMGOTTEN contains sufficient storage to meet the request, a GETMAIN is not issued; the required pages are assigned and necessary changes are made in the VMASSIGN and VMGOTTEN chains. The total amount of storage represented by entries in the VMGOTTEN chain can never exceed 255 pages.

Where it is determined that virtual storage areas represented in different entries of the same chain are adjacent, the entries are consolidated into one and the remaining block(s) freed.

SOURCE PROGRAM LISTING

The source program listing data set is created by the source program listing routine (SLLS) during Phase III, if the source listing option has been taken. It is a listing, in order, of the original source language line images submitted for assembly by the user. Warning and error messages are collected and printed at the end of the listing.

The source listing format is illustrated in Figure 87. The information contained in the listing is listed below:

- nnnnnnn
statement number, in decimal.
- SSSS-sss
source text. Terminal input greater than 120 characters is continued from column 11 of next line.
- mmmm-mmmmm
diagnostic message.

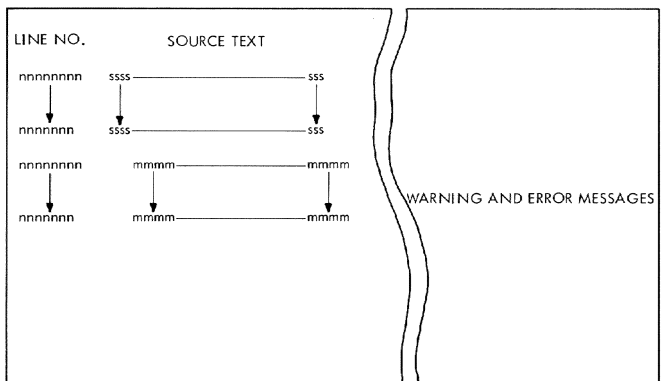


Figure 87. Source program listing format

SYMBOL	TYPE	LENGTH	VALUE	SYMBOL	TYPE	LENGTH	VALUE
ssssssss	t	11111	vv vvvvv	ssssssss	t	11111	vv vvvvv
ssssssss	t	11111	vv vvvvv	ssssssss	t	11111	vv vvvvv
ssssssss	t	11111	vv vvvvv	ssssssss	t	11111	vv vvvvv

Figure 88. Symbol table listing format

SYMBOL TABLE LISTING

The symbol table listing data set is created by the symbol table editor module (CEVSR) during Phase IV. It is a listing, in alphabetical order, of all the symbols defined within the assembly, together with their length, type, and value attributes.

The symbol table listing format is illustrated in Figure 88. The information contained in the listing is described below:

```

ssssssss
  symbol
t
  type code:
A
  A-type address constant, implied
  length, aligned.
B
  Binary constant.
C
  Character constant.
D
  long floating-point constant, implied
  length, aligned.
E
  Short floating-point constant, implied
  length, aligned.
F
  Fullword fixed-point constant, implied
  length, aligned.
G
  Fixed-point constant, explicit length.
H
  Halfword fixed-point constant, implied
  length, aligned.
K
  Floating-point constant, explicit
  length.

```

```

P
  Packed decimal constant.
Q
  Q-type address constant, implied
  length, aligned.
V
  V-type address constant, implied
  length, aligned.
X
  Hexadecimal constant.
Y
  Y-type address constant, implied
  length, aligned.
Z
  Zoned decimal constant.
R
  A-, Q-, R-, S-, V-, or Y-type address
  constant, explicit length.
S
  S-type address constant, implied
  length, aligned.
I
  Machine instruction.
J
  Control section name.
M
  Macro instruction.
T
  External symbol.
W
  CCW assembler instruction.
U
  Undefined.
#
  R-type address constant, implied
  length, aligned.

```

SYMBOL	TYPE	LNG	LOCATION	REFERENCES
ssssssss	t	11111	nn ddddd	rr ccccc, rr ccccc, rr ccccc, rr ccccc, rr ccccc, rr ccccc, rr ccccc, rr ccccc
ssssssss	t	11111	nn ddddd	rr ccccc, rr ccccc, rr ccccc, rr ccccc

Figure 89. Cross-reference listing format

11111

length of data defined by the symbol, in hexadecimal, zeros not suppressed.

vv vvvvv

value attribute, in hexadecimal

if relocatable: section number and displacement

if absolute: 8-digit number.

CROSS-REFERENCE LISTING

The cross-reference listing data set is created by the cross-reference listing processor routine (XREF) during Phase IV. It is a listing, in alphabetical order, of all symbols properly defined within the assembly, together with a list of all program locations at which a reference to the symbol is made in the source language.

The cross-reference listing format is illustrated in Figure 89. Each line specifying a symbol may have up to ten location references to the symbol. If fewer than this number of references is made to a symbol, the rest of the line is blank. If more than ten references are made, the locations making the references are listed on the next line.

The information contained in the listing is described below:

ssssssss

alphanumeric symbol, left justified

t

type code:

A

A-type address constant, implied length, aligned.

B

Binary constant.

C

Character constant.

D

Long floating-point constant, implied length, aligned.

E

Short floating-point constant, implied length, aligned.

F

Fullword fixed-point constant, implied length, aligned.

G

Fixed-point constant, explicit length.

H

Halfword fixed point constant, implied length, aligned.

K

Floating point constant, explicit length.

P

Packed decimal constant.

Q

Q-type address constant, implied length, aligned.

V

V-type address constant, implied length, aligned.

X

Hexadecimal constant.

Y

Y-type address constant, implied length, aligned.

Z

Zoned decimal constant.

R

A-, Q-, R-, S-, V-, or Y-type address constant, explicit length.

S

S-type address constant, implied length, aligned.

I Machine instruction.

J Control section name.

M Macro instruction.

T External symbol.

W CCW assembler instruction.

U Undefined.

R-type address constant, implied length, aligned.

lllll length in hexadecimal, right justified, zeros not suppressed.

nn section number location where symbol is defined, in hexadecimal.

dddd location counter displacement where symbol is defined, in hexadecimal, zeros not suppressed.

rr section number location where symbol is referenced, in hexadecimal. Blank if the symbol is not referenced. Reference locations in ascending order.

cccc location counter displacement where symbol is referenced. Blank if the symbol is not referenced.

nnnnnnnn alphameric symbol, left justified.

ttttt type:

INSTR
VALUE
SECTION
INTEGER
REAL
CHAR
HEX
BINARY
PACKED
ZONED
S-CON
ADCON

ffffff duplication factor, in hexadecimal.

lllllllll length, in hexadecimal.

dddddddd immediate value or section member and location counter displacement, in hexadecimal.

INTERNAL SYMBOL DICTIONARY (ISD) LISTING

The ISD listing data set is created by the ISD listing processor routine (ISDSA) during Phase IV. It is a listing of the

PROGRAM MODULE DICTIONARY (PMD) LISTING

The program module dictionary listing data set is created by the program module dictionary listing processor routine (PMDLS) during Phase IV. It is a listing of the external symbol definitions, references, and relocation information contained in the program module dictionary.

INTERNAL SYMBOL DICTIONARY							
NAME	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
TYPE	ttttt	ttttt	ttttt	ttttt	ttttt	ttttt	ttttt
DUPL	ffffff	ffffff	ffffff	ffffff	ffffff	ffffff	ffffff
LENGTH	lllllllll	lllllllll	lllllllll	lllllllll	lllllllll	lllllllll	lllllllll
LOC/VAL	ddddddd	ddddddd	ddddddd	ddddddd	ddddddd	ddddddd	ddddddd

Figure 90. ISD Listing Format

The program module dictionary listing format is illustrated in Figure 91. The information contained in the listing is described below:

nnnnnnnn
 alphameric name of module, section, or symbol.

xxxxxxx
 module identification.

llllllll
 length, in hexadecimal.

ccc
 severity code:
 000
 no warning or error messages.
 001
 warning message.
 002
 error message.

ss
 section number, hexadecimal.

tttttttt
 type:
 CONTROL
 COMMON
 PROTOTYPE

mm/dd/yy hh:mm:ss
 time stamp

aaaaaaaa
 attribute, one or more of following:
 VARIABLE FIXED
 READONLY SYSTEM
 PUBLIC
 PRVLGD

vvvvvvvv
 location counter displacement value, in hexadecimal.

rrrr
 reference number, in hexadecimal.

ssssssss
 DXD symbol name or (CXD) for a CXD.

llllll
 total length reserved for the symbol.

bbbbbb
 boundary alignment as BYTE, HALF, FULL, DOUBLE.

pp
 page number, in hexadecimal.

mmmm
 number of modifiers, in hexadecimal.

h
 length, in hexadecimal.
 i
 type identifier: + - C Q R
 aaa
 address, in hexadecimal.

OBJECT PROGRAM LISTING

The object program listing is created as a VISAM data set or written on SYSOUT, depending on user request and mode. This data set also contains the source program listing, cross-reference listing or symbol table listing, and PMD and ISD listings.

The object program listing shows, in control section order, the binary text assembled for each source statement. Warning and error messages are collected and presented at the end of the listing. A count of the number of messages and an indication of the highest severity code encountered are also presented.

The assembler edit feature directs a source statement to be edited in the following manner: (a) the name field will always begin in column 1; a sequence symbol in the name field is suppressed; (b) the operation code is shifted to begin in the location corresponding to card column 10 or the next available location thereafter; (c) the operand is shifted to begin in the location corresponding to card column 16 or the first available location thereafter; and (d) the comment field will follow the operand field by the number of blanks coded in the source statement. No editing is performed if the statement is in error.

Each line of the object program listing is filed as a logical record with a PUT macro or a GTWRC macro. The line image in memory is the same as it appears on the listing, except that it is preceded by one byte containing ASA FORTRAN print-spacing control information.

The listing contains the following types of lines.

- Column heading line.
- Machine instructions.
 - a. RR format
 - b. RS, RX format
 - c. SI format
 - d. SS format

MODULE									
NAME	nnnnnnnn								
MODULE ID	xxxxxxx								
LENGTH	1111111								
DIAG SEVERITY	ccc								
SECTION ss									
NAME	nnnnnnnn								
TYPE	tttttttt								
VERSION	mm/dd/yy hh:mm:ss								
ATTRIBUTES	aaaaaaaa								
CSD LENGTH	1111111								
SECT LENGTH	1111111								
RELOCATABLE DEFINITIONS									
NAME	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
VALUE	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv
NAME	hhhhhhh	hhhhhhh	hhhhhhh	hhhhhhh	hhhhhhh	hhhhhhh	hhhhhhh	hhhhhhh	hhhhhhh
VALUE	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv
ABSOLUTE DEFINITIONS									
NAME	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
VALUE	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv
COMPLEX DEFINITIONS									
NAME	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
VALUE	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv	vvvvvvvv
REFERENCES									
REF #	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr
NAME	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
DXD AND CXD REFERENCES									
REF #	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr
NAME	sssssss	sssssss	sssssss	sssssss	sssssss	sssssss	sssssss	sssssss	sssssss
LENGTH	11111	11111	11111	11111	11111	11111	11111	11111	11111
ALIGN	bbbbbb	bbbbbb	bbbbbb	bbbbbb	bbbbbb	bbbbbb	bbbbbb	bbbbbb	bbbbbb
MODIFIERS FOR COMPLEX DEFS									
PAGE pp	#	MODIFIERS	mmmm						
LENGTH	h	h	h	h	h	h	h	h	h
REF #	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr
TYPE	i	i	i	i	i	i	i	i	i
BYTE	aaa	aaa	aaa	aaa	aaa	aaa	aaa	aaa	aaa
MODIFIERS FOR TEXT (EXTERNAL REFS, Q-CONS, AND CXDS)									
TEXT PAGE pp	VIRTUAL PAGE	vv	#	MODIFIERS	mmmm				
LENGTH	h	h	h	h	h	h	h	h	h
REF #	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr
TYPE	i	i	i	i	i	i	i	i	i
BYTE	aaa	aaa	aaa	aaa	aaa	aaa	aaa	aaa	aaa
MODIFIERS FOR TEXT (INTERNAL REFS)									
TEXT PAGE pp	VIRTUAL PAGE	vv	#	MODIFIERS	mmmm				
LENGTH	h	h	h	h	h	h	h	h	h
REF #	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr
TYPE	i	i	i	i	i	i	i	i	i
BYTE	aaa	aaa	aaa	aaa	aaa	aaa	aaa	aaa	aaa
SECTION ss									
NAME									
TYPE									
VERSION									
ATTRIBUTES									
CSD LENGTH									
SECT LENGTH									
etc.									

Figure 91. Program module dictionary listing format

- Assembler instructions with related values.
- Assembler instructions without values.
- CCW Instructions.
- CNOP Instructions.
- Constants.
- Literal pools.
- Diagnostic messages.
- MNOTE Messages.
- Commentary lines.

Figures 92 and 93 illustrate the listing format.

INTERNAL SYMBOL DICTIONARY (ISD)

The assembler ISD is divided into four sections: a heading, section name table, using tables, and the symbol table. It is illustrated in Figure 94.

Heading

- Word 1
bits 0-15 contain the indicator (4) identifying the ISD as assembler produced.
- Word 2
the length of the ISD in bytes.
- Word 3
contains a link to the start of the symbol table.
- Word 4
the number of entries in the section name table.
- Word 5
the number of using tables.
- Word 6
the number of entries in the symbol table.

Section Name Table

The alphanumeric name and the version identification of each control section (including DSECTS) is entered here in sequence by the section number assigned. The name of blank common is represented by eight blank characters; the unlabeled control section is represented by binary zero.

Using Tables

The assembler places a using table in the ISD at every section break and for each USING and DROP statement. All 15 entries are included each time, plus the location at which the table became effective. Registers containing bases for DSECT references are included, but those registers containing other external bases are marked as unavailable for checkout purposes.

Symbol Table

The assembler inserts as symbol entries all absolute or simple relocatable value items from its internal dictionary, in addition to entries for each section name. Symbols are grouped according to control section and ordered within each group by ascending location counter value. Immediate value symbols follow those with location counter values.

Name	two words containing the alphameric name of the symbol.	
DSECT FLAG	contains a 1 bit if this symbol was defined in a DSECT, or if it names a DSECT.	
TYPE	identifies the type of field as:	
ISD Code	ISD Type*	Assembler Type Attributes
1	INSTR	Instruction(I)
2	VALUE	Absolute EQUs
3	SECTION	Section name(J)
4	INTEGER	Integer constant(F,H)
5	REAL	Real number(D,E)
6	CHAR	Character constant(C)
7	HEX	Hexadecimal constant (G,K,R,X,M,W,U)
8	BINARY	Binary constant(B)
9	PACKED	Packed decimal constant(P)
10	ZONED	Zoned decimal constant(Z)
11	S-CON	S-type address constant(S)
12	ADCON	Other address constant (A,Q,V,Y)

*The ISD Types show how each of the ISD codes will be displayed in the Internal Symbol Dictionary Listing.

Relocatable EQU symbols and symbols on LTORG statements will be placed in the ISD in the following manner:

1. If the statement is coded in the format

symbol EQU expression 1

LOCATION	INSTRUCTION	ADDR 1	ADDR 2	STATEMENT	SOURCE	MM/DD/YY	HH:MM:SS
	01 02710			100	ORG	10000	
	01 02710			200	USING	*,12	
01 02710	C1			300	DC	C'A'	
01 0270C	C1C2			400	DC	C'AR'	
01 02713	0123			500	ALPHA DC	X'123'	
01 02718				600	DS	0D	
01 02718	00000001			700	DC	F'1'	
01 0271C	C1C2C3C4C5C6C7C8			800	GAMMA DC	C'ABCDEFGHJIJ'	
	C9D1						
01 02726	C1C2C3C4C5			900	DC	3C'ABCDE'	
01 0272B	C1C2C3C4C5						
01 02730	C1C2C3C4C5						
01 02735				1000	DS	50CL10	
01 02929	0555			1100	DC	B'10101010101'	
01 0292B	00			+			
01 0292C	002D			1200	DC	H'45'	
01 0292E	0A			1300	DC	X'A'	
01 0292F	00			+			
01 02930	00002723			1400	DC	A (ALPHA+16)	
01 02934	00000000			1500	DC	V (RETA)	
01 02938	C003	01 02713		1600	DC	S (ALPHA)	
01 0293A	0000			+			
01 0293C	00000000			1700	DC	R (BETA)	
01 02940	271C			1800	DC	Y (GAMMA)	
01 02942	0000			+			
01 02944	00004E34			1900	DC	A (ALPHA+GAMMA+5)	
01 02948	0000000000000000			2000	DC	V (DELTA, OMEGA, SIGMA)	
	00000000						
01 02954	01239C			2100	DC	P'+1239'	
01 02957	F8F4F7F6D5			2200	DC	Z'-84765'	
01 0295C	01234C6D789C052C			2300	DC	P'1234,-6,+789,+52'	
01 02964	01D0B38DF9C			2400	DC	H'465,+15245,-100'	
01 0297A	13838188			2500	DC	FL.7'9',CL.10'AR',XL.14'C4'	
01 0297E	3AC0			2600	DC	BL.10'11101011'	
01 02970	00000009			2700	DC	F'9',C'AB',X'C4',H'-242'	
01 02974	C1C2						
01 02976	C4						
01 02977	00			+			
01 02978	FF0E						
01 0297A	0A141E			2800	DC	2HL1'10,20,30'	
01 0297D	0A141E						
01 02988	0A84150E5030A80			2900	DC	FL.10'673,21,57,259,42'	
01 02988				3000	DC	0F	
01 02988				3100	RDAREA DS	0CL80	
01 02988				3200	DS	CL4	
01 0298A				3300	PAYNO DS	CL6	
01 02990				3400	NAME DS	CL20	
01 029A5				3500	DATE DS	0CL6	
01 029A5				3600	DAY DS	CL2	
01 029A7				3700	MONTH DS	CL2	
01 029A9				3800	YEAR DS	CL2	
01 029AB	C9C2D440E261F3F6			3900	DC	CL20'IBM S/360'	
	F040404040404040						
	40404040						
				4000	END		

Figure 92. Listing format for constants

and the first term in the operand expression 1 is not an asterisk, the ISD type of the symbol will be the type of the first term. If the first term is a self-defining term, the ISD type will be HEX.

- If the first term in the operand field is a reference to the location counter (*), the ISD type will be determined

by the next statement, within the same control section, to generate text or modify the location counter.

The following cause the ISD type to be INSTR ISD code 1):

- a machine instruction
- CNOP

LOCATION	INSTRUCTION	ADDR 1	ADDR 2	STATEMNT	SOURCE
XX XXXXX	XXXX			XXXXXXXX	RR R1,R2
XX XXXXX	XXXX XXXX		XX XXXXX	XXXXXXXX	RX R1,D2 (X2,B2)
XX XXXXX	XXXX XXXX	XX XXXXX		XXXXXXXX	SI D1 (X1,B1) ,I
XX XXXXX	XXXX XXXX XXXX	XX XXXXX	XX XXXXX	XXXXXXXX	SS D1 (L,B1) ,D2 (B2)
XX XXXXX	XXXXXXXXXX XXXXXXXXX			XXXXXXXX	CCW
XX XXXXX	XX XXXXX			XXXXXXXX	CSECT (ALSO DSECT,PSECT, AND COM)
XX XXXXX	XXXX			XXXXXXXX	CNCP (NO ALIGNMENT NEEDED)
XX XXXXX	XXXX XXXX			XXXXXXXX	CNOP (HALF-WORD ALIGNMENT NEEDED)
XX XXXXX	XXXX XXXX XXXX			XXXXXXXX	CNOP (FULL-WORD ALIGNMENT NEEDED)
XX XXXXX	XXXX XXXX XXXX			XXXXXXXX	CNOP (THREE HALF-WORDS NEEDED)
	XX XXXXX			XXXXXXXX	END (INTERNAL SYMBOL)
	XXXXXXXXXX			XXXXXXXX	EQU (ABSOLUTE VALUE)
	XX XXXXX			XXXXXXXX	EQU (SIMPLE RELOCATABLE VALUE)
	XX XXXXX			XXXXXXXX	ORG (ALSO LTRG)
	XXXXXXXXXX			XXXXXXXX	USING (ABSOLUTE VALUE)
	XX XXXXX			XXXXXXXX	USING (RELOCATABLE VALUE)
	XXXXXXXXXX			XXXXXXXX	SETA (ALSO SETB)
	AAAAAAAA			XXXXXXXX	SETC
				+ PLUS SIGN INDICATES GENERATED LINE
				E E INDICATES ERROR
				XXXXXXXX THIS FORMAT IS USED FOR ALL OTHER ASSEMBLER INSTRUCTIONS

Figure 93. Listing format for machine and assembler instructions

The following cause the ISD type to be determined as if it were a constant:

- a. DC
- b. DS
- c. CXD

The following cause the ISD type to be HEX (ISD code 7):

- a. ORG
- b. END
- c. EQU
- d. CCW
- e. LTRG

The following instructions will not affect the ISD type because they do not modify the location counter or generate text:

- a. PRINT
- b. SPACE
- c. EJECT
- e. USING
- f. DROP
- g. CSECT
- h. PSECT
- i. DSECT
- j. COM
- k. DXD
- l. a macro statement
- m. any statement in a macro definition (between MACRO and MEND)

- n. COPY
- o. SETA, SETB, SETC
- p. LCLA, LCLB, LCLC
- q. GBLA, GBLB, GBLC
- r. ANOP
- s. AGO
- t. AIF
- u. ENTRY
- v. EXTRN
- w. MNOTE
- x. comment(*)

If the user has specified the type in the third operand field of the EQU statement, the ISD type will be assigned as follows:

Hex Type	ISD Type
00-7A	HEX
7B	ADCON
7C-C0	HEX
C1	ADCON
C2	BINARY
C3	CHAR
C4-C5	REAL
C6	INTEGER
C7	HEX
C8	INTEGER
C9	INSTR
CA-D6	HEX
D7	PACKED
D8	ADCON
D9-E1	HEX
E2	S-COM
E3-E4	HEX
E5	ADCON
E6-E7	HEX
E8	ADCON
E9	ZONED
EA-FF	HEX

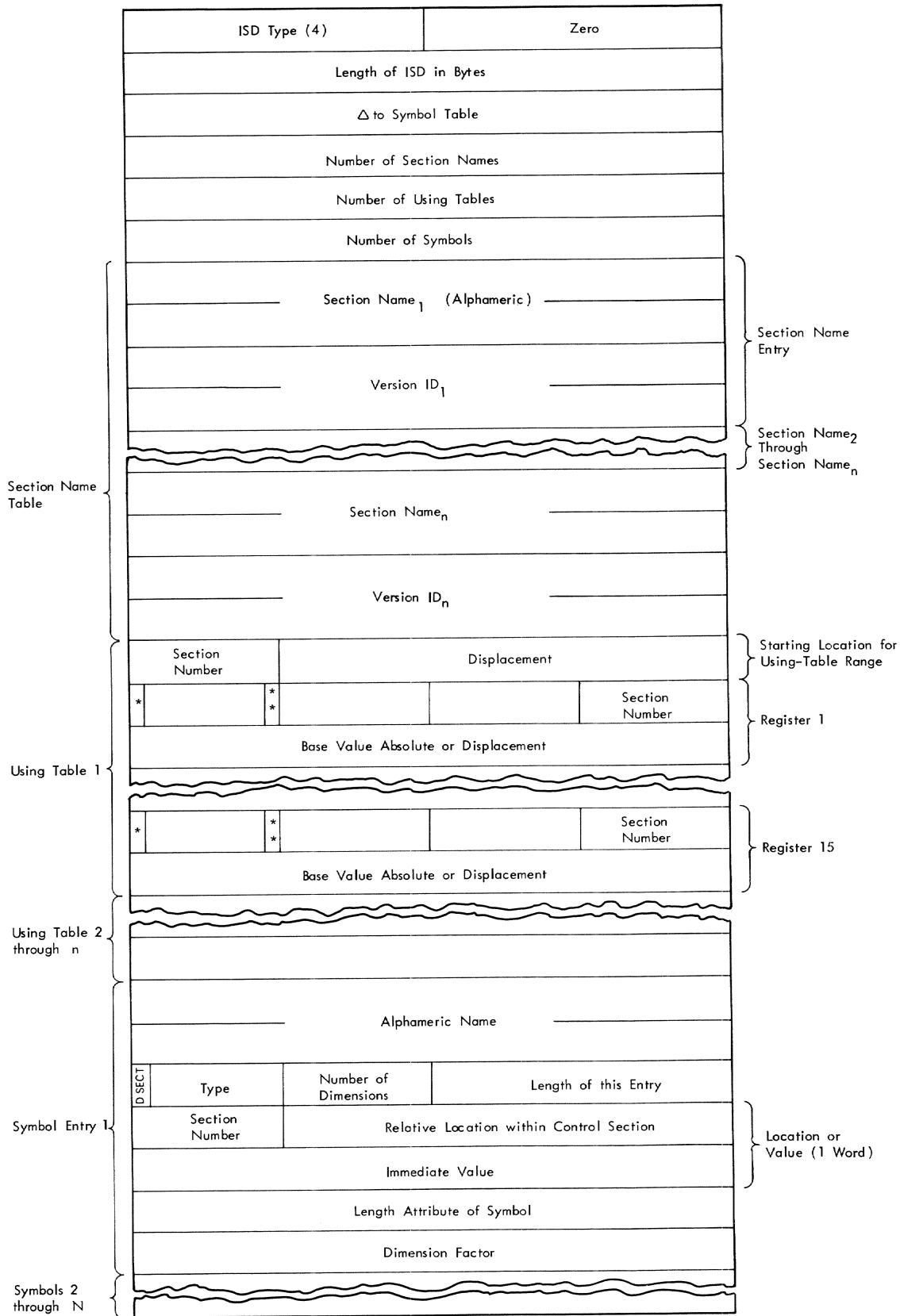


Figure 94. Assembler internal symbol dictionary

The ISD type will be HEX if there is any conflict between the type and the alignment of the symbol.

Symbols with type attributes T,N, and O are not included in the ISD, nor do undefined symbols appear.

Number of dimensions has a value of 1 if a duplication factor (other than 1) or multiple contents were used. Otherwise, it has a value of 0.

Length of entry length in bytes for this symbol entry.

Section number a number identifying the section the symbol was defined in. This corresponds to the ordering of the names in the section name table.

Displacement the location counter value.

Immediate value if the type was indicated as an absolute EQU, the fourth word of the symbol entry will contain, instead of a section number and displacement, the immediate value of the symbol.

Length length in bytes of the field defined by this entry.

Dimension factor this word is included in the symbol entry only if the number of dimensions is nonzero. It contains the byte length of the entire field defined by this entry (i.e., the length times the duplication factor).

PROGRAM MODULE DICTIONARY (PMD)

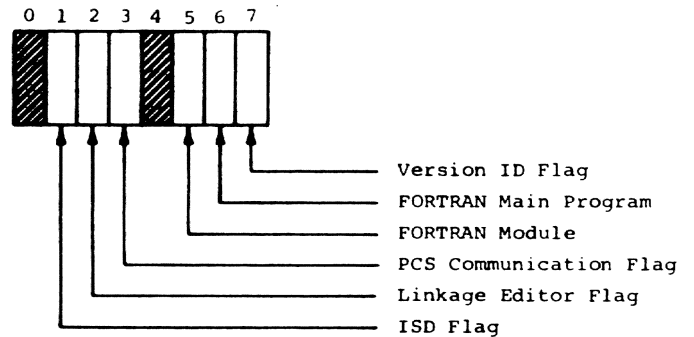
Each PMD consists of one PMD heading plus as many control section dictionaries (CSD) as there are control sections in the module (excluding DSECTS) Address pointers in the PMD are initially relative to the beginning of the PMD itself (not the PMD preface), except where otherwise specified. Some fields in the PMD are filled in by the loader. These are left zero by the language processor. The PMD format is shown in Figure 95.

PMD Heading

1. Length of PMD in bytes.
This length does not include the PMD preface.
2. Diagnostic code (1 byte).
The diagnostic code indicates the

highest level diagnostic encountered during generation of the module by the language processor that created it.

3. Flags (1 byte).
The flag bits are numbered from left to right starting with zero and are defined as follows:



Bit 1 module has an ISD associated. This bit is set by the processor creating the PMD.

Bit 2 module was processed by link editing. This bit is set by the Linkage Editor.

Bit 3 PCS is to be called before module is dynamically unlinked. This bit is set by PCS.

Bit 5 module was produced by the FORTRAN compiler.

Bit 6 FORTRAN module is a main program, not a SUBROUTINE, FUNCTION, or BLOCK DATA subprogram.

Bit 7 version ID indicator. If this bit is set, the module version ID is to be interpreted as a 64-bit binary number which is the creation date of the module. If this bit is not set, the version ID is eight alphanumeric EBCDIC characters.

4. Length of PMD heading.

This is the length in bytes of the PMD heading.

5. 4-character I.D. name.

The 4-character I.D. name is supplied by the user to serve as deck identification if the module is punched into cards. This field is currently unused.

The PMD Preface is Prefixed here by either STARTUP or the Dynamic Loader.

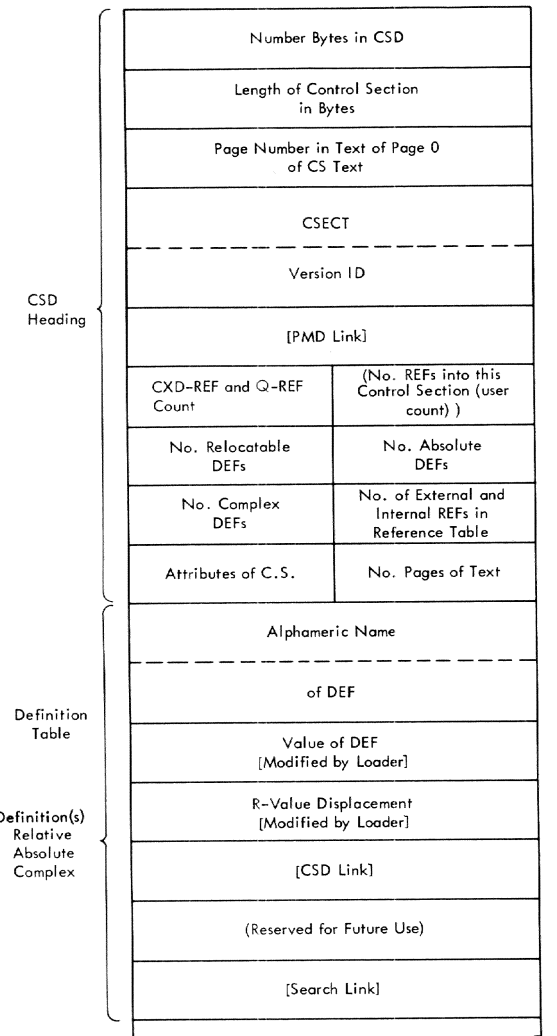
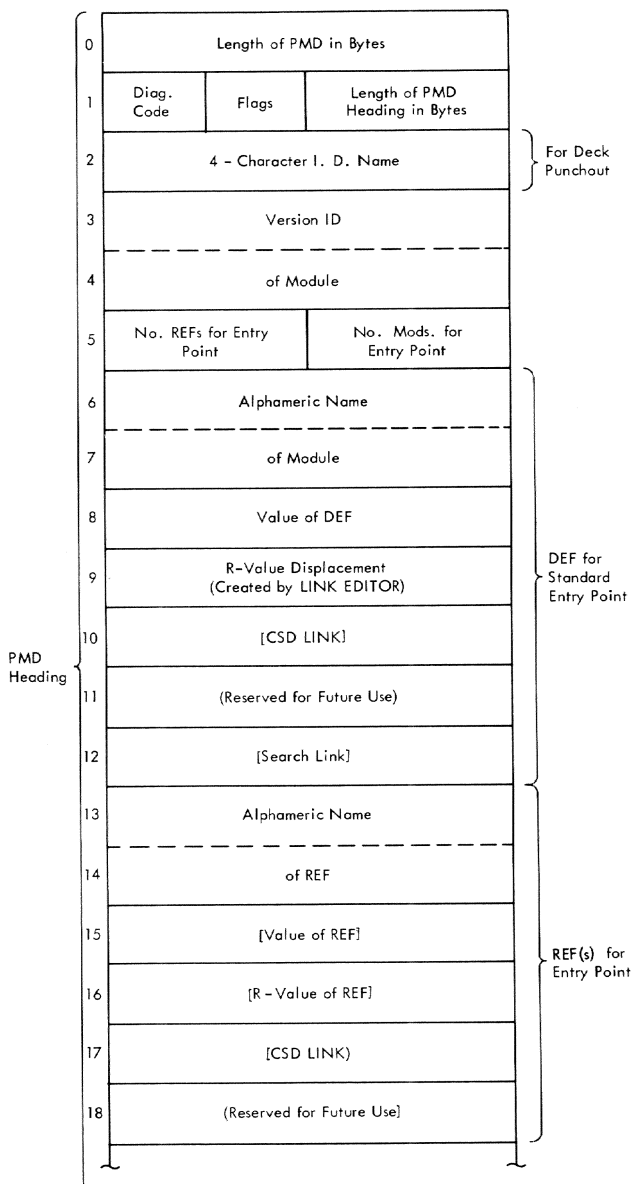
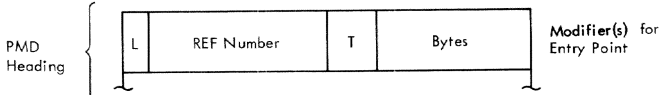
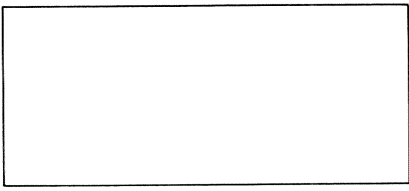


Figure 95. Program module dictionary entry format (part 1 of 2)

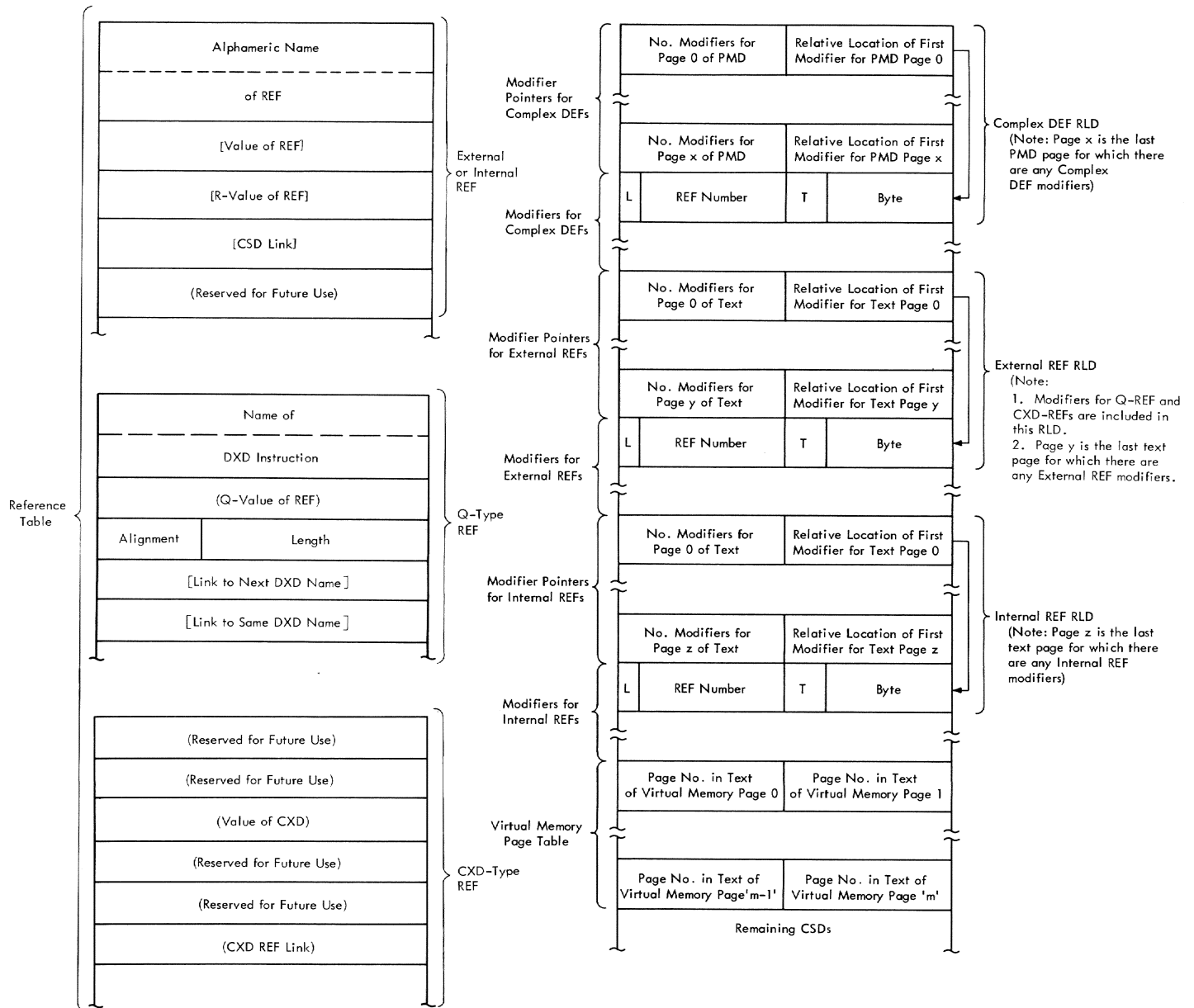


Figure 95. Program module dictionary entry format (part 2 of 2)

6. Version I.D.

See item 3 above (Bit 7 discussion) for interpretation of version I.D.

7. Number of REFS for the standard entry point.

The DEF for the standard entry point is always treated as a complex DEF. This field contains the number of REFS. It may be zero.

8. Number of modifiers for the standard entry point.

This field contains the number of modifiers that are to be used to compute the DEF for the standard entry point.

9. DEF for standard entry point.

This 7-word entry describes the DEF for the standard entry point of the module. It has the same form as the individual DEF entries within the CSDs. The standard entry point DEF for the module is considered to belong to the first PSECT of the module and is treated the same as a complex DEF

whose ENTRY statement appears within that PSECT. If no PSECT is declared, the standard entry point will be associated with the first CSECT instead. This DEF entry contains the following subfields:

- a. Alphameric name of module
- b. Value of DEF
- c. R-value displacement
- d. CSD link
- e. Reserved for future use
- f. Search link

The alphameric name is also the name of the module.

10. REF(s) for entry point - These have the same form and function as the REFs described in the CSD discussion below.
11. Modifier(s) for entry point - These have the same form and function as the modifiers for the RLD for complex definitions described in the CSD discussion below, except that they apply to the standard entry point DEF.

CONTROL SECTION DICTIONARY (CSD)

The control section dictionary has the following components:

1. CSD Heading
2. Definition table
3. Reference table
4. Relocation dictionaries (RLDs)
5. Virtual memory page table (VMPT)

CSD HEADING

1. Number bytes in CSD - This field specifies the length of the control section dictionary in bytes.
2. Length of control section in bytes - This specifies the virtual memory span of the control section. The length of the virtual memory page table is derived from this length. For example, if the length of the control section is 8192, the virtual memory page table will contain two pages; but if the length is 8193 bytes, the virtual memory page table will contain three pages. This value will be equal to the highest location counter value assigned by the language processor, plus one.

3. Page number in text of page 0 of CSECT text - The text for each control section in the module occupies an integral number of pages in its resident data set. The text pages for all control sections in a module are contiguous. This number is the page number, relative to the first page of text for this module, of the first page of text for this CSECT. (Numbering begins with 0.)
4. Version I.D. - This is a 64-bit binary number which is the creation date of the control section expressed as the number of microseconds that have elapsed from March 1, 1900, until the time of CSECT creation. This number is changed by the linkage editor when CSECT combining occurs.
5. PMD link - The PMD is filled in by STARTUP or the dynamic loader. It points to the beginning of the PMD preface.
6. Whether CSD-type REF and number of Q-type REFs.

Bits from left to right contain:

- a. Bit 0 - set to 0 if no CXD-type REF exists; set to 1 if a CXD-type REF exists. (Only one CXD-type REF is possible.)
- b. Bit 1 - not used.
- c. Bits 2-14 - number of Q-type REFs. (Contains all zeros if none.)
7. Number of implicit references to this control section (user count) - This is a count of the number of REF entries that refer to this control section and are linked to this CSD through their CSD link. It is computed by the loader. It includes both external and internal references. This number is arbitrarily set by STARTUP for each CSECT in initial virtual memory to X'7FFF' to prevent unloading of IVM modules.
8. Number of relocatable definitions - This is the number of relocatable definitions in the definition table. It is always at least one, namely, the control section name DEF.
9. Number of absolute definitions - This is the number of absolute definitions in the definition table. It may be zero.
10. Number of complex definitions - This is the number of complex definitions in the definition table. It may be zero.

11. Number of references from this CSD - This is the sum of external and internal references in the reference table. It may be zero.
12. Attributes - This halfword has one bit set for each attribute possessed by the control section. Currently defined attributes are shown below.

Bits are numbered from left to right starting with 0.

- a. Fixed-length (Bit 14 off) - A fixed-length control section is a section of fixed length. It will be allocated a fixed number of pages at load time.
- b. Variable-length (Bit 14 on) - A variable-length control section is of indeterminate length. It will be allocated pages in excess of the length stated in the CSD heading.
- c. Read-only (Bit 13 on) - Read-only specifies that no data can be stored in the control section. Causes memory protection by means of a storage class-B assignment to all pages of the control section. Nonread-only and nonprivileged CSECTs are assigned storage class A.
- d. Public (Bit 12 on) - Control sections are not shared by CSECT name alone. A public control section of a module residing in a given data set (library) is shared if another user has access to the same data set and module. CSECTs of a given module need not all be public or non-public. Fixed-length public CSECTs with the same attributes are assigned storage in the same assignment. A public CSECT must never contain relocatable adcons (A, V, or R type).
- e. PSECT (Bit 11 on) - If this bit is set, it causes the dynamic loader to override the system packing indicator and insert this control section as packed.
- f. Common (Bit 10) - A common section is a control section common to all modules in which it is declared. Common sections are more fully discussed in the Linkage Editor and Assembler Language SRLs.

Common sections are of two types:

- (1) Named common sections (those with a name not all blanks). These are treated as fixed-length sections.

- (2) Blank common sections, whose name consists of eight blanks. FORTRAN blank common is assigned the variable and common attributes by the FORTRAN compiler.

The treatment of blank common sections differs from that of blank non common sections. Control section rejection is instituted between blank common sections of different modules whereas blank non-common sections of different modules are treated as independent control sections. The latter are called unnamed control sections.

- g. Privileged (Bit 9 on) - A CSECT with a privileged attribute is assigned storage key C which provides fetch as well as store protect. This attribute overrides R/O. Anything in a privileged CSECT may be referenced only when the PSW key is zero.
- h. System (Bit 8 on) - Any external symbol that appears in a CSECT with the system attribute cannot be referenced by a user program unless the symbol begins with "SYS". Conversely, no reference from a control section with a system attribute may be to a "user" symbol.
- i. TDYCQR validity (Bit 7 on) - The language processor sets this flag to indicate that the count of Q-type REFS in TDYCQR is valid. If bit 7 is off, the count of Q-type REFS is not valid.
- j. Common CSECT rejected (Bit 6 on) - The dynamic loader sets this flag to indicate to the Program Control System that the CSECT was rejected as a common CSECT that was already loaded in another module.
- k. Bits 4 and 5 are not used.
- l. Public storage assigned by CONNECT (CZCGA7) (Bit 3 on) - Set by the dynamic loader if applies.
- m. PCSA (CGCCT) called for this CSD (Bit 2 on) - Set by the dynamic loader if applies.
- n. CSD has been allocated storage (Bit 1 on) - Set by the dynamic loader if applies.
- o. Public name (Bit 0 on) - This is used only by the dynamic loader to specify nonblank control sections whose names appear in the SDST (Shared Data Set Table). The

first such control section will appear in the SDST under the module name. A section may be indicated as both having a public name and rejected.

13. Number of Pages of Text - This specifies the number of pages of text for this control section in the data set. It should be noted that this generally does not correspond to the number of pages in the virtual memory page table. It cannot be larger.

DEFINITION TABLE

The definition table is made up of 7-word entries, one for each external definition in the current control section. Definitions are grouped as relocatable, absolute, and complex in that order. The first definition in the table is the name of the current control section.

Relocatable definitions are external definitions whose value may be computed as the sum of the origin of the control section wherein they appear, and a constant that is the symbol's displacement from the section origin.

An absolute definition is an EQU item with an absolute value whose name has been declared an entry point in the CSECT in which the name is defined.

A complex definition is either an EQU item with a complex relocatable value (that is, containing external symbols) or a simple relocatable definition whose ENTRY statement appeared within a control section other than the section in which it is defined. The definition entry appears within the CSD of the control section that contains the ENTRY statement. (Note that the origin of the same control section is the R-value for the DEF.) The complex DEF is required in this case, with one REF entry that names the control section in which the DEF symbol is actually defined.

Each DEF in the definition table contains the following entries:

1. Alphameric name of DEF - This field contains the 8-character alphameric name of the DEF.
2. Value of DEF - The value of the DEF is set by the language processor and is modified by STARTUP or the loader in the case of complex and relocatable definitions. For relocatable DEFs, the value portion of the definition entry contains the displacement value of the symbol relative to the base of its control section. For absolute

DEFs, this entry contains the absolute value; for complex DEFs it contains the absolute portion of the DEF value, which may be zero.

3. R-Value displacement - The "displacement for R-value" word contains the displacement of the original defining control section origin with respect to the head of the control section within which the definition now appears. This is required to compute valid R-values for control sections which have been combined by linkage editing. In creating the PMD, only the linkage editor will ever produce a nonzero value in this word.
4. CSD link - The CSD link is initially zero. It is filled in by STARTUP or the dynamic loader when the control section is loaded as a pointer to the beginning of the CSD in which this DEF appears, provided that neither the DEF nor the control section has been rejected.
5. Reserved for future use.
6. Search link - This field is filled by the HASH SEARCH routine of either the loader or STARTUP. It contains the address of the beginning of the next DEF entry, which hashes to the same value. It contains zero if there are no more DEFs with the same hash value in this chain.

REFERENCE TABLE

The reference table is made up of 6-word entries, one for each external symbol referenced within the control section. Each entry for an external or internal REF contains the following:

1. Alphameric name of REF - This field contains the 8-character alphameric name of the REF.
2. Value of REF - This is filled in by STARTUP or the dynamic loader. It contains the value of the DEF to which the REF refers. If the DEF is undefined, it contains the address of a portion of virtual memory wherein reference is illegal.
3. R-Value of REF - This is filled in by STARTUP or the dynamic loader. It contains the virtual memory address of the beginning of the control section wherein the DEF appears. This value is obtained from the "R-value displacement" word of the satisfying DEF entry.

If the DEF is undefined, this word contains the address of a portion of virtual memory wherein reference is illegal.

4. CSD link - This pointer, initially zero, is filled by STARTUP or the dynamic loader. It points to the beginning of the CSD wherein the DEF that defines this REF appears. If a corresponding DEF could not be found upon the appearance of a REF, the CSD link is to the beginning of the CSD wherein the REF itself appears.
5. Reserved for future use.

Each entry for a Q-type REF contains:

1. Name of DXD instruction - The eight-character alphanumeric name of a DXD instruction, or a DSECT name referenced in a Q-type address constant.
2. Q-value of REF - This is filled in by the RESOLVE QREF routine of the dynamic loader. It contains the displacement from the beginning of the combined dummy sections of the dummy section defined by the DXD instruction.
3. Alignment, Length - The alignment and length specified by the assembler language processor.
4. Link to next DXD name - This is filled in by the Q-CHAIN routine of the dynamic loader when Q-CHAIN posts the REF on one of the eleven hash chains for Q-type REFs.
5. Link to same DXD name - This is filled in by the Q-CHAIN routine of the dynamic loader when Q-CHAIN posts the REF on one of the secondary Q-type REF chains for duplicate-name DXDs.

Each entry for a CXD-type REF contains:

1. For future use.
2. Value of CXD - This is filled in by the EXPLICIT LINK routine of the dynamic loader. It contains the length of the combined dummy sections.
3. For future use.
4. CXD REF link - This is filled in by the ALLOCATE MODULE routine of the dynamic loader as CXD-type REFs are chained together.

RELOCATION DICTIONARY (RLD)

Three RLDs appear in each control section dictionary. The three RLDs are:

1. RLD for complex definitions
2. RLD for internal references
3. RLD for external references

Each RLD has the same format consisting of modifier pointers and modifiers. The RLD for complex definitions differs in that pages mentioned in this table are pages of the PMD rather than the text.

Modifier Pointer

Modifier pointers are used to designate the application of modifiers to adcons on appropriate pages of text (or of the PMD for complex defs). The first modifier pointer applies to the first page; the second modifier pointer, the second page; etc. For an RLD there always exists at least one modifier pointer. However, there need not necessarily be a modifier pointer for each page of text; the modifier pointers may be ended at the last text page for which there exists any modifier.

The modifier pointers consist of two fields, in the left and right halfwords.

Left half

Number of modifiers for page - This field contains the number of modifiers that apply in this page.

Right half

Location of first modifier for this page - This contains the location in bytes relative to the right half of the pointer itself for the first modifier for this page. If there are none, it points to the location where one would have appeared if there were any.

A special note should be made of the technique for determining the length of the RLD. If one looks in the right half of the first pointer for an RLD, one finds the location of the first modifier for this page. In the word preceding the first modifier word is the last modifier pointer for the RLD. By adding the location of the right half (of the last pointer) to the contents of the right half (of the last pointer), one gets the beginning of the last set of modifiers. Adding to this four times the number of modifiers in the last set, one gets the end of the RLD.

Modifier

1. L - L (2 bits) is the length in bytes of the adcon to be modified. A value of zero indicates a fullword (4 bytes).

2. Ref number - Reference number (14 bits) is the ordinal number in this CSD's reference table of the reference whose definition value is to be used in modifying the adcon. References are numbered starting with zero.
3. T - T (4 bits) is the operation to be performed in modifying the adcon by the reference value. The values of T currently defined are as follows:
 - a. Addition (T = 1) - The definition value of the reference at "Reference Number" is added to the field of L bytes at the location specified by "Byte".
 - b. Subtraction (T = 2) - Same as addition, except read "subtracted from" for "added to."
 - c. R-value (T = 3) - The "R-value" of the REF is stored into the field of length L at the location specified by "Byte".
 - d. Q-value (T = 4) - The Q-value of the REF is stored into the field of length L at the location specified by "Byte".
 - e. Value of CXD (T = 5) - The value of the CXD instruction is stored into the field of length L at the location specified by "Byte".
4. Byte - Byte (12 bits) is the displacement in bytes (from the origin of its original containing page) of the adcon to be modified. It should be noted that since PMDs are packed to word boundaries, this displacement will be added to an address for complex DEFs which generally is not a page boundary.

RLD for Complex Definitions

The format of these modifiers is as described above. These modifiers apply to the values of complex definitions; that is, the byte addresses in the modifier will be added to the value words of complex DEF entries in the definition table, and the page numbers in the modifier pointers are for pages of the program module dictionary itself.

RLD for Text External Reference

This relocation dictionary is in the same form as described above. It has one pointer for each page of program text up to that text page, which is the last to contain an adcon, and appropriate modifiers for each adcon in the text, which refers to

a symbol defined externally to this module. The page numbers are based on the first page for this control section, beginning with 0.

RLD for Text Internal Reference

This is identical to RLD for text external reference above, except that the modifiers are to adcons in the text which reference symbols defined within this module, such as control section names. This permits communication between control sections of the same module that may be allocated noncontiguous virtual memory.

VIRTUAL MEMORY PAGE TABLE (VMPT)

This table has a halfword for each page of virtual memory that the CSECT occupies, beginning with page 0 and continuing upward in order.

The contents of each entry will be either:

1. All one bits if the corresponding page is empty as a result of a DS or ORG statement.
2. The number of the page in the text relative to the beginning of text for this CS if the page contains code or data.

This table is the means by which the text of the control section is related to the virtual memory assigned the control section. This is because language processors do not necessarily output a byte of text for each byte of virtual memory assigned; that is, large ORG and DS statements may result in pages of text being skipped.

If, for example, a source program were to begin with

```
ORG      10000
```

there would be no text output for the first two pages of virtual memory, and the first page of text would correspond to the third page of the user's virtual memory. The first two VMPT entries would be all bits, and the third would contain zero. Within a page, however, the bytes of text correspond directly to the bytes of virtual memory. Thus, in the example above, the first page of text would represent virtual memory locations 8192-12,287, and the first 1808 bytes of the page of text would be vacant (10,000-8192 = 1808). The pages of text will always begin on page boundaries within the text module.

SECTION 13: VIRTUAL MEMORY MANAGEMENT

PURPOSE OF VIRTUAL MEMORY MANAGEMENT ROUTINES

Virtual memory management is performed by three routines: VMGET, VMFREE, and VMCLEAN. These routines provide the assembler with efficient virtual storage allocation services. Whenever working storage is required or no longer needed by the assembler, these routines are called. They keep track of virtual storage obtained from the system and determine when to issue GETMAIN and FREEMAIN macro instructions on behalf of the assembler. The virtual memory management routines are part of assembler module CEVAL.

Assemblies of unusually large source programs may require an increase in the amount of storage requested with each GETMAIN; privileged system programmers (authority code 0) may change the storage requested by altering certain constants. See "Changing Storage Request Constants" at the end of this section.

HOW VIRTUAL MEMORY MANAGEMENT WORKS

The virtual memory management routines maintain a common table (named VMTABLE) containing blocks that describe virtual storage extents obtained from the system via the GETMAIN macro instruction. By consulting and maintaining entries in VMTABLE blocks, VMGET determines whether to allocate to a requesting phase of the assembler storage already obtained or whether to request new storage by issuing a GETMAIN. VMFREE maintains entries in VMTABLE of storage no longer needed by the assembler. VMCLEAN, at termination of the assembly, consults entries made by VMGET and VMFREE and issues required FREEMAIN macro instructions. (These routines are described individually below.)

VMTABLE itself contains 64 three-word blocks; each block is a member of one of four chains within the table. The chain headers, VMGOTTEN, VMASSIGN, VMFREED, and VMENTRYS, are address constants outside VMTABLE; each points to the first block of its chain within VMTABLE. The chain headed by VMGOTTEN records areas obtained via GETMAIN and not in present use by the assembler. The chain headed by VMASSIGN records areas obtained via GETMAIN and presently assigned to some phase of the assembler. The chain headed by VMFREED records areas previously acquired via GETMAIN, relinquished by some phase of the assembler, and

now available for return to the system via FREEMAIN. The chain headed by VMENTRYS simply points to unused blocks within VMTABLE.

The format and contents of VMTABLE blocks are shown in Section 12 under "Virtual Memory Management Table."

ROUTINES

VMGET -- Get VM Working Storage (CEVGM)

This routine issues GETMAIN macro instructions for the assembler. GETMAIN requests are made as seldom as possible; they are not issued when previously secured storage is still available. (See Chart EA.)

Entry Point: CEVGM

Calling Sequence: L R0,length
LA R1,option
INVOKE ACEVGM

(length = number of pages requested
option: 1 = 'EXIT=RETURN', 0 = 'NO RETURN')

Routines Called: None

Macro instruction: GETMAIN

Routines That Call VMGET:

PHASE I (Phase I Master Control)
REED
COPY
MACREF
PHASE III (Control)
ISDPR
SSCAN

Exit:

Normal - R1 contains address of first virtual memory page.
R15 contains a return code:

0 - normal
4 - insufficient virtual storage and 'EXIT=RETURN'

Error - ABEND 1 - 'CEVGM-GETMAIN REQUEST OF 0 PAGES'.
ABEND 2 - 'CEVGM-TOO MANY VM REQUESTS FROM ASSEMBLER' (VMTABLE overrun).
ABEND 2 - 'CEVGM-VM EXHAUSTED' (and no 'EXIT=RETURN').

OPERATION: VMGET either secures the requested storage from pages represented by VMGOTTEN entries or, if there are not

enough unassigned pages available to satisfy the request, issues a GETMAIN. At least one segment (256 pages) is requested with each GETMAIN; if less than a segment was requested by the caller, VMGET requests a segment. Amounts greater than 256 pages are requested as specified to VMGET in register 0. (Ordinarily, however, calls to VMGET will request fewer than 256 pages.)

If a request for zero pages is encountered, VMGET issues an ABEND (completion code 1). Where insufficient virtual storage is available, VMGET issues an ABEND or returns to the caller, depending on the EXIT setting in register 1.

After successful completion of GETMAIN or when enough pages are available to satisfy the request without a GETMAIN, VMTABLE is altered to show the pages put in use, and VMGET returns with a return code of 0.

VMFREE -- Free VM Working Storage (CEVFM)

This routine accepts FREEMAIN requests from other assembler routines and records in the VMFREED chain of VMTABLE the delimiting addresses for the pages to be freed. VMFREE does not call FREEMAIN; rather, VMCLEAN uses the entries in the VMFREED chain to call FREEMAIN. VMFREE edits the contents of the VMFREED chain so that as few FREEMAINS as possible need to be issued by VMCLEAN. (See Chart EB.)

Entry Points: CEVFM
FM450 (internal to CEVA1)

Calling Sequence: L R0,length
L R1,address

INVOKE ACEVFM
(length = number of pages to be freed
address = location of first page to be freed)

Routines Called: None

Routines That Call VMFREE:

AC (Assembler Control)
PHASE I (Phase I Master Control)
PHASE III (Control)
CSDPR
ISDPR
VMCLEAN (at FM450)

Exit:
Normal - BR 14

Error - ABEND 1 - 'CEVFM-ATTEMPT TO FREEMAIN ADDRESS 0'.
ABEND 1 - 'CEVFM-ATTEMPT TO FREEMAIN LENGTH ZERO'.
ABEND 1 - 'CEVFM-ATTEMPT TO FREE UNASSIGNED VM'.
ABEND 1 - 'CEVFM-TOO MANY VM

REQUESTS FOR ASSEMBLER' (No more blocks left in VMTABLE).

OPERATION: Upon entry, VMFREE determines whether an attempt has been made to free address 0, length 0, or unassigned virtual storage. If any of these is detected, VMFREE issues an ABEND with completion code 1 to end the assembly.

VMFREE next determines whether the FREEMAIN request is for the entire range given in the appropriate VMASIGN chain entry. If this is the case, VMFREE removes that block from the VMASIGN chain and either places the block on the VMFREED chain or, if possible, makes the range of pages freed part of an existing VMFREED entry. VMFREE then places the old VMASIGN block on the available block (VMENTRYS) chain. Note that when several blocks of contiguous pages are freed, VMFREE detects this and produces only one VMFREED entry.

If the FREEMAIN request was for only part of the pages represented in a VMASIGN entry, VMFREE changes the VMASIGN chain to reflect the deletion and changes the VMFREED chain to show the newly freed pages.

VMCLEAN -- Assembler Cleanup (CEVCU)

This routine frees working storage areas obtained via GETMAIN. At termination of an assembly, VMCLEAN operates in one of two modes: normal end or early end. In early end mode, all areas represented in the VMGOTTEN, VMASIGN, and VMFREED chains are freed. In normal end mode, only areas represented in the VMGOTTEN and VMFREED chains are freed. Areas represented in the VMASIGN chain at this time represent the output module's PMD, text, ISD, and external names list. All these items are referenced by LPC (Language Processor Control), which must free the areas itself. (See Chart EC.)

Entry Point: CEVCU

Calling Sequence: LA R1,option
INVOKE ACEVCU
(option: 0=normal end, nonzero=early end)

Routines Called:

VMFREE (at FM450) - to put VMGOTTEN and VMASIGN entries on VMFREED chain (non-standard linkage)

Macro instruction: FREEMAIN

Routines That Call VMCLEAN: AC (Master Control)

Exit: Normal - BR R14

OPERATION: For both normal and early end modes, VMCLEAN transfers blocks on the VMGOTTEN chain to the VMFREED chain by converting each entry from the VMGOTTEN format to the VMFREED format and then calling VMFREE (at FM450) to put the block on the VMFREED chain.

In addition, in early end mode (which means the assembly did not complete and no program module will be produced), VMCLEAN calls VMFREE to put each VMASIGN entry onto the VMFREED chain. In normal mode, the areas represented by VMASIGN entries will not be returned to the system immediately; they must be saved for reference by Language Processor Control in producing a program module.

In either mode, unless the VMFREED chain is empty, VMCLEAN issues a FREEMAIN macro instruction to free the pages represented by each entry in the VMFREED chain.

Before returning to the caller, in either mode, VMCLEAN places all blocks (including VMASIGN blocks) onto the VMENTRYS (available blocks) chain, so that VMTABLE will be initialized for later assemblies during the same task.

CHANGING STORAGE REQUEST CONSTANTS

Each routine that requests virtual storage through a call to the virtual memory management routines specifies its own constant number of pages. For an unusually large assembly, one or more of these constants may be insufficient and a work area may overflow. (Examples: the assembly of a user's own language processor, or a SYSGEN assembly of the TSS/360 system.) Programmers with authority code O

may "tune" these constants to meet their needs.

The constants that may be altered, the work areas for which they are used to requisition storage, the normal value of the constants (number of pages), and the usual reasons for overflow are shown in Table 12.

CAUTION

Since these constants reside in PUBLIC, READONLY code, they must not be altered if any other user is assembling.

The constants must be altered prior to an assembly for which they will be used. Altering constants during assembly (via the attention key, then a GO or RUN command after the alteration) will usually cause an ABEND.

OVERFLOW DIAGNOSIS

The assembler will attempt to dynamically obtain additional virtual storage should a work area overflow. However, if storage is unavailable or unable to be addressed, the assembly is terminated with a diagnostic message naming the work area which overflowed. A knowledge of PMD or ISD control block sizes may be helpful when analyzing PMD or ISD work area overflows. The CHATDY and CHAISD DSECTS, listed in System Control Blocks, GY28-2011, describe components of the PMD and ISD, respectively, and may aid in determining proper altered values for CEVPMD and CEVISD.

Additional information on resetting constants is available in System Programmer's Guide, GC28-2008 under "Tuning the TSS/360 Assembler."

Table 12. Virtual storage request constants

Name of Constant	Work Area	Normal Value	Possible Reasons for Overflow and Comments
CEVW1	Work 1	100 pages	Too many continued source lines. Too many USING or DROP statements. Insufficient room for 2-word cross-reference items. Nesting of macro calls causes generation of macro level dictionaries requiring too much space.
CEVW2	Work 2	255 pages	Too many symbols in name fields. Too many source statements. Too many macro-generated statements. Too many continued lines. Insufficient work space for building control section dictionary.
CEVW3	Work 3	20 pages per increment (more as needed to hold source statements)	Seldom occurs. More than 50,000 lines of code (including source, library or assembly macro definition, and generated statements may cause VMTABLE to overflow. Increase the size of constant CEVW3.
CEVXL	External Name List Area (EXT NAM)	2 pages + size of the DEFs (number of DEFs multiplied by 28)	Number of control sections and ENTRY operands exceed 1022.
CEVPM D	PMD Work Area	2 pages + (number of text pages / 8) + size of the DEFs and REFS (the total number of DEFs and REFS multiplied by 28)	Too many ENTRY or EXTRN operands with little or no text for module. (Text = executable instructions + constants).
CEVISD	ISD Work Area	0 pages + number of pages in Work 2	The value of CEVW2 (size of Work 2) too small to contain ISD. Too much USING/DROP information and symbolic name information for work area.

APPENDIX A: ASSEMBLER REGISTER USAGE

The general purpose machine registers are coded symbolically throughout the assembler and are used as follows:

- R0 - Parameter register (effectively volatile)
- R1 - Parameter register (effectively volatile)
- R2 - Parameter register (effectively volatile)
- R3 - Parameter register (effectively volatile)
- R4 - Parameter register (effectively nonvolatile)
- R5 - Parameter register (effectively nonvolatile)
- R6 - Volatile
- R7 - Volatile
- R8 - Permanently addresses working area 1 (nonvolatile)
- R9 - Permanently addresses read-only constants (nonvolatile)
- R10 - Permanently addresses working area 2 (nonvolatile)
- R11 - Used for local addressability by individual subroutines (nonvolatile)
- R12 - Nonvolatile
- R13 - Permanently addresses Assembler's PSECT
- R14 - Linkage register
- R15 - Linkage register

APPENDIX B: RELATIONSHIP OF DOCUMENTATION MODULES TO ASSEMBLY MODULES

For purposes of checkout, system integration, and maintenance, the modules documented in the internal specifications have been collected into fifteen assembly modules. The list below indicates the relationship between system integration assembly modules and the corresponding documentation.

<u>Assembly Module</u>	<u>Documentation Modules</u>
CEVA1 (Phase I, Part I)	CEVMR (Machine Operations Requirements Table) CEVDX (Diagnostic Message Processor) CEVBS (Basic Scan Routine) CEVKM (Lookup Dictionary Item) CEVLP (Dictionary Lookup and Put) CEVAC (Assembler Control) CEVGW (Interface with VISAM PUT or GTWRC Macro) CEVGM (Get VM Working Storage) CEVFM (Free VM Working Storage) CEVCU (Assembler Cleanup)
CEVA2 (Phase I, Part II)	CEVCS (Constant Scan Routine)
CEVA3 (Phase I, Part III)	CEVEV (Expression Evaluator) CEVGB (Binary Self-defining Term Generator) CEVGC (Character Self-defining Term Generator) CEVGD (Decimal Self-defining Term Generator) CEVGH (Hexadecimal Self-defining Term Generator) CEVPS (Parameter Item Analyzer) CEVTM (Macro Name Item Insert) CEVLM (Macro Name Item Lookup) CEVTK (Macro Dictionary Lookup) CEVTP (Macro Dictionary Insert)
CEVA4 (Phase I, Part IV)	CEVRD (Obtain Next Source Statement) CEVSS (String Substitution Routine) CEVSP (Substitute into Operation Code)
CEVA5 (Phase I, Part V)	CEVOP (Operation Code table) CEVGP (Identify Operation Code) CEVCP (Substitution Control Routine) CEVST (Statement Analyzer) CEVGO (AGO/AIF Instruction Scan) CEVAN (ANOP Instruction Scan) CEVCW (CCW Instruction Scan) CEVCT (Control Section Instruction Scan)
CEVA6 (Phase I, Part VI)	CEVCN (CNOP Instruction Scan) CEVCY (COPY Instruction Scan) CEVCX (CXD Instruction Scan) CEVDC (DC/DS Instruction Scan) CEVEJ (EJECT Instruction Scan) CEVND (END Instruction Scan) CEVEY (ENTRY Instruction Scan) CEVXN (EXTRN Instruction Scan) CEVQU (EQU Instruction Scan) CEVGL (Global/Local Symbol Scan) CEVIC (ICTL Instruction Scan) CEVIQ (ISEQ Instruction Scan) CEVLG (LTOrg Instruction Scan) CEVMX (MEND/MEXIT Instruction Scan) CEVMN (MNOTE Instruction Scan)
CEVA7 (Phase I, Part VII)	CEVMP (Machine Instruction Scan) CEVMC (MACRO Instruction Scan) CEVRG (ORG Instruction Scan)

	CEVPH (PUNCH Instruction Scan)
	CEVPR (PRINT Instruction Scan)
	CEVSE (SET Statement Scan)
	CEVCE (SPACE Instruction Scan)
	CEVTI (TITLE Instruction Scan)
	CEVUD (USING/DROP Instruction Scan)
	CEVRE (REPRO Instruction Scan)
CEVA8 (Phase I, Part VIII)	CEVRF (Macro Reference Processor)
	CEVDF (Macro Definition Processor)
	CEVSL (Literal Operand Scan)
	CEVSY (Define Location Symbol)
CEVA9 (Phase I, Part IX)	CEVPA (Phase I Control)
CEVB1 (Phase IIA)	CEVPB (Phase IIA Control)
	CEVPM (Macro Parameter Processor)
CEVC1 (Phase IIB)	CEVPC (Phase IIB Control)
	CEVEQ (Assign Value to Name Routine)
	CEVRL (Literal Resolution Processor)
	CEVGN (Location Counter Reset Routine)
	CEVLC (Location Counter Assignment Routine)
	CEVPL (Literal Pooling Processor)
	CEVRS (Alignment Resolution Routine)
CEVD1 (Phase IIC)	CEVPD (Phase IIC Control)
	CEVUP (USING Table Processor)
	CEVDR (DROP Statement Processor)
CEVE1 (Phase III, Part I)	CEVPE (Phase III Control)
	CEVMO (Machine Operations Processor)
	CEVUV (Using-Register Routine)
	CEVLT (Literal Text Processor)
	CEVEP (Entry Point Processor)
	CEVCC (CCW Text Processor)
CEVE2 (Phase III, Part II)	CEVGV (Get Relocatable Value Routine)
	CEVPV (Output Relocatable Value Routine)
	CEVAD (Address Constant Processor)
	CEVLS (Object Program Listing Routine)
	CEVDP (DC Constant Processor)
	CEVCD (CSD Processor)
	CEVSX (Source Listing Processor)
CEVF1 (Phase IV)	CEVPF (Phase IV Control)
	CEVXF (Cross-Reference Listing Routine)
	CEVSR (Symbol Table Edit Routine)
	CEVSD (ISD Processor)
	CEVMD (PMD Listing Routine)
	CEVSA (ISD Listing Routine)

OBJECT PROGRAM

PMD

The amount of virtual storage required to hold the PMD is computed, then allocated via a GETMAIN macro. A limitation on the size of the PMD would occur only if the system were unable to allocate sufficient pages to hold the computed size of the PMD.

TEXT

Text is allocated in the same way as the PMD. It is limited only by virtual storage allocation constraints.

ISD

As many pages are procured for the ISD as are allocated for the symbol table. Since the ISD is a condensation of the symbol table, it appears that any symbol table which can be assembled can be placed in a corresponding ISD.

SOURCE STATEMENTS

The pages allocated for the symbol table and associated working space allow a total program size of approximately 30,000 source statements. Each statement uses a minimum of four words in an encoded list: if the statement defines a symbol, it will use a minimum of about five words; if the statement defines a constant, it will use a minimum of another four words. The 30,000 figure is derived by assuming that 20 per-

cent of the source statements of a program are constants which define symbols, and that 29 percent of the remaining statements define symbols. Most classes of statements are not numerically limited per se, if the statement can be encoded within the allocated working storage. There is, however, a limit of 256 different control sections (regardless of type), 256 LTOrgs, and $2^{16}-1$ external references.

MACROS

There is no restriction on the number of different macros provided both the definition statements and the expansions can be contained within the 30,000 statement limit. (Each macro name counts as one symbol in the table.) Nor is there a particular limit on the number of usages, so long as all the expanded statements can be contained within the previously mentioned limit. However, &SYSNDX is only a four-place counter and will become ambiguous after 10,000 macro calls.

A single macro call operand is limited to 255 characters. The total number of positional operands is limited to 255.

Maximum Statement Length

By language definitions, card format is limited on ordinary statements to one card and two continuations: 240 characters. Macro calls and macro prototypes are effectively without limit, as are keyboard format statements, so long as the system can allocate space to the assembler to hold the images.

APPENDIX D: ACRONYMS

AADCON	Fullword pointer to entry point for ADCON
AAGO	Fullword pointer to entry point in AGO/AIF for AGO instructions
AAIF	Fullword pointer to entry point in AGO/AIF for AIF instructions
AANOP	Fullword pointer to entry point for ANOP
ABSCAN	Fullword pointer to entry point for BASCAN
ACATOP	Fullword pointer to entry point for CATOP
ACCW	Fullword pointer to entry point for CCW
ACCWTX	Fullword pointer to entry point for CCWTXT
ACEVCS	Fullword pointer to entry point for CXD
ACEVKM	Equated to ADLKM
ACEVLM	Fullword pointer to entry point for MACLKT
ACEVOP	Fullword pointer to beginning of working copy of operation code table
ACEVPA	Fullword pointer to entry point for CEVPA
ACEVPB	Fullword pointer to entry point for CEVPB
ACEVPC	Fullword pointer to entry point for CEVPC
ACEVPD	Fullword pointer to entry point for CEVPD
ACEVPE	Fullword pointer to entry point for CEVPE
ACEVPF	Fullword pointer to entry point for CEVPF
ACEVSR	Fullword pointer to entry point for STED
ACEVTM	Fullword pointer to entry point for MACPUT
ACNOP	Fullword pointer to entry point for CNOP
ACONST	Fullword pointer to beginning of constants
ACOPY	Fullword pointer to entry point for COPY
ACSCAN	Fullword pointer to entry point for CSCAN
ACSCOM	Fullword pointer to entry point in SECT for COM instructions
ACSDCT	Fullword pointer to entry point in SECT for DSECT instructions
ACSDPR	Fullword pointer to entry point for CSDPR
ACSECT	Fullword pointer to entry point in SECT for CSECT instructions
ACSPCT	Fullword pointer to entry point in SECT for PSECT instructions
ACSTRT	Fullword pointer to entry point in SECT for START instructions
ACTSD	Fullword pointer to current temporary dictionary
ADB1	Address of address constants used in calling GETLINE
ADC	Fullword pointer to entry point in DC/DS for DC instructions
ADC1	Address of address constants used in calling PUTDIAG
ADCON	Equated to beginning of address constants for the individual processors for machine and assembler instructions used during Phases I and IIA
ADCSW	One byte used by Phase IIB to indicate where ADCON literals are to be pooled
ADCTXT	Fullword pointer to entry point for DCTXT
ADD	Constant of X'F4'
ADFSYM	Fullword pointer to entry point for DEFSYM
ADIAG	Fullword pointer to entry point for DIAG
ADLKM	Fullword pointer to entry point for DLKM
ADLKT	Fullword pointer to entry point for DLKT
ADL	Fullword pointer to beginning of diagnostic locator table
ADLNG	Fullword used to indicate length of dummy RLD created by ADCON for V-type address constant
ADLPM	Fullword pointer to entry point for DLPM
ADM	Fullword pointer to beginning of table of diagnostic messages
ADOOP	Fullword pointer to entry point for subroutine internal to EVAL
ADPUT	Fullword pointer to entry point for DPUT
ADRLD	Fullword pointer to dummy RLD created by ADCON for V-type address constant

ADROP	Fullword pointer to entry point in USE/DROP for DROP instructions
ADRSET	Fullword pointer to entry point for DRSET
ADS	Fullword pointer to entry point in DC/DS for DS instructions
ADSCAN	Fullword pointer to entry point for SSCAN
ADTVF	One byte used to indicate when the value of an address constant has been truncated
AEATT	Fullword pointer to entry point for subroutine internal to EVAL
AEBIN	Fullword pointer to entry point for EBIN
AECHAR	Fullword pointer to entry point for ECHAR
AEDEC	Fullword pointer to entry point for EDEC
AEFWD	Fullword pointer to entry point for subroutine internal to EVAL
AEHEX	Fullword pointer to entry point for EHEX
AEJECT	Fullword pointer to entry point for EJECT
AEND	Fullword pointer to entry point for END
AENDPR	Fullword pointer to entry point for ENDPR
AENTRY	Fullword pointer to entry point for ENTRY
AEQATE	Fullword pointer to entry point for EQUATE
AEQU	Fullword pointer to entry point for EQU
AEVAL	Fullword pointer to entry point for EVAL
AEVLG	Fullword pointer to entry point for subroutine internal to EVAL
AEXTRN	Fullword pointer to entry point for EXTRN
AGATEW	Fullword pointer to processor that interfaces with PUT VISAM macro
AGBLA	Fullword pointer to entry point in GBLX/LCLX for GBLA instructions
AGBLB	Fullword pointer to entry point in GBLX/LCLX for GBLB instructions
AGBLC	Fullword pointer to entry point in GBLX/LCLX for GBLC instructions
AGETOP	Fullword pointer to entry point for GETOP
AGLOC	Fullword pointer to entry point for GBLX/LCLX common to all GBLX and LCLX instructions
AGTVAL	Fullword pointer to entry point for GETVAL
AICTL	Fullword pointer to entry point for ICTL
AISDLS	Fullword pointer to entry point for ISDSA
AISDPR	Fullword pointer to entry point for ISDPR
AISEQ	Fullword pointer to entry point for ISEQ
ALCATE	Fullword pointer to entry point for LOCATE
ALCLA	Fullword pointer to entry point for GBLX/LCLX for LCLA instructions
ALCLB	Fullword pointer to entry point in GBLX/LCLX for LCLB instructions
ALCLC	Fullword pointer to entry point for GBLX/LCLX for LCLC instructions
ALINE	Fullword indicating antepenultimate source line number
ALIST	Fullword pointer to entry point for LIST
ALIT	Fullword pointer to entry point for subroutine internal to EVAL
ALITXT	Fullword pointer to entry point for LITXT
ALMHSB	Equated to AMNHSB
ALOCTR	Fullword pointer to second word preceding the location counter value
ALTLN	Fullword used to indicate where page-usage table was constructed
ALTORG	Fullword pointer to entry point for LTORG
AMACRO	Fullword pointer to entry point for MACRO
AMCDEF	Fullword pointer to entry point for MACDEF
AMCREF	Fullword pointer to normal entry point in MACREF
AMEND	Fullword pointer to entry point in MEND/MEXIT for MEND instruction
AMEXIT	Fullword pointer to entry point in MEND/MEXIT for MEXIT instruction
AMHSB	Fullword pointer to main dictionary hash table
AMNHSB	Fullword pointer to macro name dictionary hash table

AMNOTE Fullword pointer to entry point for MNOTE
 AMOPR Fullword pointer to entry point for MOPR
 AMOPRQ Fullword pointer to beginning of machine operation require-
 ments table
 AOPCDE Fullword pointer to beginning of permanent copy of operation
 code table
 AOPCD1 Fullword pointer to end of permanent copy of operation code
 table
 AOPRR1 Fullword pointer to entry point in MIP for RR1 machine
 instructions
 AOPRR2 Fullword pointer to entry point in MIP for RR2 machine
 instructions
 AOPRR3 Fullword pointer to entry point in MIP for RR3 machine
 instructions
 AOPRS1 Fullword pointer to entry point in MIP for RS1 machine
 instructions
 AOPRS2 Fullword pointer to entry point in MIP for RS2 machine
 instructions
 AOPRX1 Fullword pointer to entry point in MIP for RX1 machine
 instructions
 AOPRX2 Fullword pointer to entry point in MIP for RX2 machine
 instructions
 AOPSI1 Fullword pointer to entry point in MIP for SI1 machine
 instructions
 AOPSI2 Fullword pointer to entry point in MIP for SI2 machine
 instructions
 AOPSS1 Fullword pointer to entry point in MIP for SS1 machine
 instructions
 AOPSS2 Fullword pointer to entry point in MIP for SS2 machine
 instructions
 AOP999 Fullword pointer to end of working copy of operation code
 table
 AORG Fullword pointer to entry point for ORG
 AORGIN Fullword pointer to entry point for ORIGIN
 APARAM Fullword pointer to entry point for PARAMAC
 APMDLS Fullword pointer to entry point for PMDLS
 APOLIT Fullword pointer to entry point for POOLIT
 APRINT Fullword pointer to entry point for PRINT
 APSCAN Fullword pointer to entry point for PSCAN
 APTVAL Fullword pointer to entry point for PUTVAL
 APUNCH Fullword pointer to entry point for PUNCH
 APZ002 Fullword pointer to place where Phase IIA returns control
 when Phase IIA processing is completed
 APZ003 Fullword pointer to place where DIAG and REED pass control
 when PUTDIAG or GETLINE returns with an abnormal end code
 APZ008 Fullword pointer to place where Phase IV returns control once
 Phase IV processing is completed
 ARDPL Fullword pointer to parameter list supplied to GETLINE
 AREED Fullword pointer to entry point for REED
 AREPRO Fullword pointer to entry point for REPRO
 ARLD Fullword pointer to STAK2
 ARSCON Fullword pointer to main entry point in RESCON
 ARSLIT Fullword pointer to entry point for RESLIT
 ARS100 Fullword pointer to entry point in RESCON for byte alignment
 ARS870 Fullword pointer to entry point in RESCON for halfword
 alignment
 ASETA Fullword pointer to entry point in SETX for SETA instructions
 ASETB Fullword pointer to entry point in SETX for SETB instructions
 ASETC Fullword pointer to entry point in SETX for SETC instructions
 ASLIT Fullword pointer to entry point for SLIT
 ASLLS Fullword pointer to entry point for SLLS
 ASPACE Fullword pointer to entry point for SPACE
 ASSPTR Fullword pointer to beginning of STACK
 ASTAN Fullword pointer to entry point of STAN
 ASTANV Fullword pointer to beginning of standard variable table
 AST090 Fullword pointer to place where STAN calls REED to obtain the
 next line
 ASUBOP Fullword pointer to entry point for SUBOP

ATDFLT	Fullword pointer to default attributes
ATHSH	Fullword pointer to temporary dictionary hash table
ATITLE	Fullword pointer to entry point for TITLE
ATRANS	Fullword pointer to transitive item chain
AUSET	Fullword pointer to entry point for USET
AUSING	Fullword pointer to entry point in USE/DROP for USING instructions
AUSVAL	Fullword pointer to entry point for USEVAL
AWORK1	Fullword pointer to address of available core in working segment 1
AWORK2	Fullword pointer to address of available core in working segment 2
AWORK3	Fullword pointer to address of available core in working segment 3
AXREF	Fullword pointer to entry point for XREF
BASCCS	Fullword pointer to base of text module for current control section
BCOL	Fullword used to indicate begin column (card format)
BCOL2	Fullword save area for begin column indicator
BORC	One byte used to indicate whether current assembly is being completed in batch or conversational mode
BSBEG	Fullword pointer to field last scanned
BSCHAR	One byte used by BASCAN as code to indicate type of character found in one-byte scan
BSERR	One byte set by BASCAN to indicate what type of error was found
BSLNTH	Full word which contains length of field last scanned
BSMAX	Fullword pointer to end of current statement
BSMI	One byte set by BASCAN to indicate the type of delimiter encountered
BSMOVE	Fullword used to indicate in which storage area a character string is to be constructed
BSSCAN	Fullword pointer to where scanning is to begin
BSTAT	Equated to beginning of status area
BSTEMP	One-byte of temporary storage for translated character used by BASCAN
BSTYPE	One byte set by BASCAN to indicate the contents of the scanned field
CCOL	Fullword used to indicate continue column (card format)
CCOL2	Fullword save area for continue column indicator
CCSPAG	Fullword used to indicate the virtual memory page number associated with the first page of text for the current control section
CCS	Fullword pointer to dictionary item associated with control section currently in control
CDWORK	Two-word work area used by CSDPR
CEVBLO	Fullword save area used by PARAMAC
CEVBOO	Fullword save area used by PARAMAC
CEVIBY	Fullword save area used by PARAMAC. Referenced indirectly through STM instruction
CEVLCI	Fullword pointer to constant item
CEVLIT	Fullword save area used by PARAMAC
CEVLOB	Fullword used to indicate binary text location in bits
CEVLOT	Fullword used to indicate binary text location in bytes
CEVOTL	Fullword used to indicate the length of binary text output
CEVSW	One byte used to indicate when first duplication has been processed
CEVWA	Fullword save area used by PARAMAC
CLC	Fullword which contains the value of the current location counter
CLI	Command language interpreter
CNVFLD	510-byte area into which BASCAN puts a converted character string
CSACC	Three-word area used as an accumulator (CSCAN)

CSBIT Constant of X'10000000'
 CSBTLN Fullword indicating bit length of constant (CSCAN)
 CSCHAR One byte used by CSCAN as save area for a translated character
 CSCLOF One byte used by CSCAN to indicate the LOF directive code of the instruction being processed
 CSDCHI Character constant of 2147483647
 CSDUP Fullword used to indicate duplication factor (CSCAN)
 CSD Control section dictionary
 CSEND Fullword pointer to end of current statement (CSLOC+CSEND)
 CSERR One byte set by CSCAN to indicate if an error has occurred
 CSEXP Fullword used to indicate exponent modifier (CSCAN)
 CSEXPB Fullword pointer to exponent digits (CSCAN)
 CSEXPL Fullword used to indicate the number of exponent digits (CSCAN)
 CSEXPS Fullword used to indicate sign of exponent (CSCAN)
 CSFRB Fullword pointer to fractional part of value (CSCAN)
 CSFRBT One word that contains the number of bits in excess of the number of bytes specified in the length of the DC or DS instruction
 CSFRL Fullword used to indicate the number of fractional digits (CSCAN)
 CSINTB Fullword pointer to integer digits (CSCAN)
 CSINTL Fullword used to indicate number of digits (CSCAN)
 CSINTS Fullword used to indicate sign of value (CSCAN)
 CSLMOD Fullword used to indicate length modifier (CSCAN)
 CSLNG Fullword that contains the length of the current statement
 CSLOC Fullword pointer to the location of the current statement
 CSLOCR Fullword used to indicate if the location counter has been referenced
 CSLOG2 Halfword constant of the log of 10 to base 2
 CSLSP Fullword constant of 524288
 CSMPCT One byte used by CSCAN to indicate if multiple constants are being specified
 CSNPAG Fullword used to indicate the number of pages needed to hold the text in the current control section
 CSOPN Fullword pointer to operand of current statement
 CSSCAL Fullword used to indicate scale modifier (CSCAN)
 CSSF One word that contains the floating-point working scale factor
 CSSLDA Shift instruction which is object of EXECUTE instruction
 CSSLDL Shift instruction which is object of EXECUTE instruction
 CSSLL Shift instruction which is object of EXECUTE instruction
 CSSLMT Two-word area in which upper- and lower-scale limits for F-, H-, D-, or E-type constants are saved
 CSSRDL Shift instruction which is object of EXECUTE instruction
 CSSTAD One word that contains the displacement of constant value from beginning of value field in constant item
 CSSW One byte used by CSCAN to indicate whether F-, H-, E-, D-, Z-, or P-type constant is being processed
 CSTEMP Four-word temporary storage area used by CSCAN
 CSTRUC One byte used to indicate when value must be truncated
 CSTYPE One byte used by CSCAN to indicate the type of constant being processed
 CSWA 48-byte area in which the values of fixed- and floating-point constants are put into format
 CSWK2 Equated to CSWA+44
 CSWL One byte used to indicate if floating- or fixed-point numbers must be truncated
 CSWORK Doubleword work area (CSCAN)
 CTATTR One-byte code set by SECT to indicate the attributes of the control section being processed
 CTCNT One byte used to indicate the number of different control sections
 CTCODE One-byte code set by SECT to indicate the type of control section being processed
 CTERSW One-byte code used by SECT to indicate when error condition exists

CURDGN Fullword pointer to last diagnostic LOF processed during Phase III

CURGSM Fullword pointer to current GSM entry

CURLOF Fullword pointer to LOF for statement currently being processed (points to GP050 when LOF is being created)

CURVMP Fullword that contains the number of the virtual memory pages currently in use

CXDREF One byte flag used to indicate the number of CXD instructions present in the control section. Zero indicates none, non-zero indicates one or more CXD instructions

CYNAME Doubleword that contains the symbol indicating what is to be copied from the library

CYNUMB Fullword save area for source statement number associated with the COPY statement

C00U1 Equated to TDFLT

C11 Fullword constant of 11

C12 Fullword constant of 12

C13 Fullword constant of 13

C14 Fullword constant of 14

C15 Fullword constant of 15

C16 Fullword constant of 16

C17 Fullword constant of 17

C20 Fullword constant of 20

C28 Fullword constant of 28

C31 Fullword constant of 31

C32 Fullword constant of 32

C40 Fullword constant of 40

C48 Fullword constant of 48

C64 Fullword constant of 64

C80 Fullword constant of 80

C100 Fullword constant of 100

C128 Fullword constant of 128

C255 Fullword constant of 255

C256 Fullword constant of 256

C400 Fullword constant of 400

C1021 Fullword constant of 1021

C2048 Fullword constant of 2048

C4096 Fullword constant of 4096

C21474 Constant of X'000002147483647F' -- maximum value of a decimal self-defining term

DC800 One byte used by DC/DS processor to indicate whether a DC or DS is being processed

DMCNT Fullword which contains a count of the diagnostic messages

DN Equated to BSMI

DPCTR Fullword used as save area for the amount specified as the duplication factor less 1

DPIC Fullword used to indicate the bit length of a constant

DPLOF Fullword pointer to current LOF entry as maintained by DCTXT

DPTR Fullword pointer to a dictionary item

DROPSW One byte set by USE/DROP to indicate whether USING or DROP is being processed

DUMMYLOF Four-word area in WORK2 set up as a comment LOF. Pointed to by the GSM entry for the first LOF if the LOF requires a corresponding GSM entry

DX001 132-character area in which diagnostic message is formatted when in conversational assembly and phases

DX002 Fullword pointer to DX003

DX003 Address of parameter list supplied to PUTDIAG

DX004 Fullword used to indicate the length of the diagnostic message being issued

DX009 Constant of X'38000000'

ECOL Fullword used to indicate end column (card format)

ECOL2 Fullword save area for end column indicator

EIGHT Fullword constant of 8

ENDCCS Fullword pointer to end of text module for current control section
 ENDIND One byte used to indicate whether or not an end statement or end of file has been encountered
 ENDLOF Fullword pointer to LOF entry for END statement
 ENDWK3 Fullword pointer to the end of working segment 3
 ERRIND Table of error and warning indicators associated with diagnostic messages
 EVTERM1 Four bytes indicating location of first term
 EVTERM1L One byte indicating length, in characters, of EVTERM1:
 00 - not set
 FF - first term is one of the following (all of which should have length attribute of one): I',L',S',T', self-defining term, location counter reference
 other - the length in character
 EXTNAM Fullword pointer to the beginning of the external name list

FA One byte used as a code to indicate the type of term encountered in left operand field
 FAL One byte used to indicate whether expression is a logical or arithmetic expression
 FAZ One byte used to indicate phase of assembler currently in control
 FB One byte used as a code to indicate the type of term encountered in right operand field
 FCPL One byte used as a count of the parenthesis levels that have occurred
 FEX One byte used to indicate the type of expression evaluated by the expression evaluator
 FFT One byte used by EVAL to indicate when the first term of an expression is being evaluated
 FIVE Fullword constant of 5
 FN One byte used as code to indicate the type of the next term
 FOUR Fullword constant of 4
 FPL One byte used to prevent multiple diagnostic messages being issued once the maximum number of parenthesis levels has been exceeded
 FRA One byte used to indicate whether term is absolute or relocatable
 FSTCON Fullword pointer to the first control section dictionary item
 FSTDGN Fullword pointer to diagnostic LOF processed during Phase III
 FSTGSM Fullword pointer to beginning of GSM chain
 FSTLIT Fullword pointer to first unpooled literal
 FSTLOF Fullword pointer to beginning of logical order file
 FSTPCT Fullword pointer to the first PSECT dictionary item
 FSTQREF Fullword pointer to the first Q-REF temporary RLD item
 FSUBE One byte used to indicate the type of expression encountered at a given parenthesis level

GLINC Equated to ZGLOC+24 -- fullword used to indicate difference between lengths of base dictionary items for subscripted and unsubscripted symbol (GBLX/LCLX)
 GLLNTH Equated to ZGLOC+20 -- fullword used as save area for length of the base dictionary item for the instruction being processed (GBLX/LCLX)
 GLOCCD One byte set by GBLX/LCLX to indicate the type of instruction being processed
 GLOCSW One byte set by GLOC to indicate whether a global or local instruction is being processed
 GMSG One byte used to indicate to LPC whether a diagnostic is to be typed out immediately or saved until the next GETLINE is issued
 GP050 Six-word area where LOF entry is created
 GP051 One byte used to indicate the length of the LOF entry for the current instruction
 GP070 Fullword pointer to entry point in SYSINDEX

GP071 Address of parameter list supplied to SYSINDEX
 GP072 Address of T-type address constant used in call to SYSINDEX
 GP990 19-word save area used when GETOP calls macro retrieval processor
 GSM Global-section-macro chain -- three-word entries pointing to statements that receive special processing in Phases IIA, IIB, IIC, and III

 ICTLER One byte set by ICTL to indicate whether column 80 has been specified as the end column
 ICTLSW One byte used to indicate when an address constant that references * is contained in a literal
 IMD Nine-word save area used by ISD listing processor
 INTATR Halfword which indicates integer attribute of last constant evaluated
 ISD Internal symbol dictionary

 LCOL One byte used to indicate leftmost column of field to be sequence checked
 LISTM One byte used to indicate that a statement is in a blank COM, DSECT, or MACRO definition, and that no text is associated with the statement
 LITLN An area of 41 halfwords in which the dummy source statement for a literal is put into format
 LITLNF Two halfwords used to indicate the source line number associated with a given literal
 LITSW One byte used to indicate whether a literal is being scanned
 LOCTR Address of pseudo dictionary item for current location counter
 LOFLK Fullword save area for 20-bit pointer to the macro instruction LOF
 LOF Logical order file -- encoded entries used to describe source statements
 LPC Language processor control
 LSBUF 136-byte area in which lines to be output by the object listing processor are put into format
 LSCDFLG One byte flag indicating current listing line, (a) card format, (b) length of source=80, (c) ECOL=71
 LSCTR Fullword used by LIST to indicate the number of lines that have been used on the current listing page
 LSDX One byte used to indicate the diagnostic flag associated with the current statement
 LSHADR Fullword pointer to object listing subheading
 LSIQB Fullword pointer to next I/O buffer
 LSPGN Fullword used to indicate the number of pages that have been listed
 LSSLNG Fullword used to indicate the length of the current source statement
 LSSLOC Fullword pointer to current source statement
 LSSPC Fullword used to indicate the amount to be spaced as specified in a SPACE instruction
 LSTADR Five-word parameter list that is supplied to the macro retrieval processor
 LSTAT Equated to length of status area
 LSTLNG Fullword used to indicate the number of bytes in the binary text
 LSTQREF Fullword pointer to the last Q REF temporary RLD item
 LSTLLL Fullword used to indicate the length in bytes of title in a TITLE instruction
 LSTTLN Fullword pointer to title indicated by TITLE instruction
 LSTXTN Fullword pointer to last item included in external dictionary item chain
 LS005 Two doublewords used as work area in conversions
 LS077 Fullword used as save area by LIST
 LTHATR Halfword which indicates length attribute of last constant evaluated
 LTRGNO One byte used to indicate the number associated with an LTRG

MCLIB	One byte used to indicate when a macro library is available
MDLIST	Four words in which parameter list is built for PMD listing processor
MDS	Macro definition switch -- one byte used to indicate whether a macro definition is being processed
MDSV	19-word save area used by PMD listing processor
MLVL	Fullword which indicates the current macro level
MODBAS	Fullword pointer to beginning of text module
MODNAM	Two-word save area for the module name
MONNE	Equated to MOZ32
MSCASL	Fullword pointer to argument scan used by GET argument -- a subroutine internal to PARAMAC
MSCCSA	Fullword pointer to current sub-argument
MSCE	One byte used to indicate error in label
MSCEOA	One byte used to indicate when the last argument has been encountered
MSCERR	One byte used to indicate when an invalid keyword has been encountered
MSCFAS	Fullword pointer to beginning of argument string
MSCKB	Fullword pointer to current keyword entry
MSCKLC	Fullword pointer to first keyword entry
MSCKSW	One byte used to indicate when the first macro or prototype operand has been encountered
MSCKWA	One byte used to indicate when a keyword has been encountered
MSCKWD	Two-word save area for keyword symbol % prefixed by &<
MSCL	Fullword containing length of prototype label
MSCLC	Fullword used as &SYSLIST counter
MSCLOF	Fullword pointer to LOF entry for macro instruction
MSCM	One word used as parenthesis level counter
MSCME	Fullword pointer to end of macro instruction line
MSCMG	Fullword which indicates length of macro instruction line
MSCML	Fullword pointer to beginning of macro instruction line
MSCMO	Fullword pointer to beginning of macro instruction operand
MSCMOD	One byte used to indicate whether macro instruction or prototype operand is being scanned
MSCP	Fullword pointer to prototype line
MSCPA	Fullword pointer to &SYSLIST entry being constructed
MSCPBP	Fullword pointer to current &SYSLIST entry
MSCPE	Fullword pointer to end of prototype line
MSCPG	Fullword which indicates the length of prototype line
MSCPLC	Fullword pointer to first &SYSLIST entry
MSCPLL	Fullword used to indicate the length of the symbol in the name field of a prototype line
MSCPO	Fullword pointer to beginning of prototype operand 1
MSCPSW	One byte used to indicate when first legal positional operand has been encountered
MSCR6	Fullword register save area
MSCR14	Fullword register save area
MSCR66	Fullword register save area
MSCS	Fullword pointer to parameter item skeleton for prototype operand
MSCSAS	Fullword pointer to current character position
MSCSBA	One byte used to indicate presence of sub-argument
MSCSBW	Fullword pointer to beginning of sub-argument
MSCSDT	One byte used to indicate presence of self-defining term
MSCSF	One byte used to indicate presence of legal symbol or nonsymbol
MSCWA	Fullword used as register save area
MSCWK1	Fullword save area for value in AWOR1
MSCX	Fullword used as &SYSNDX counter
MSC14	Fullword register save area used by PARAMAC when calling DIAG
MSC14A	Fullword register save area
MSC4	Fullword register save area used when calling DIAG
M0Z12	Constant of X'FFF00000'
M0Z29	Constant of X'FFFFFFF8'
M0Z30	Constant of X'FFFFFFFC'
M0Z32	Constant of X'FFFFFFF'
M1Z31	Constant of X'7FFFFFFF'

M12Z8 Constant of X'000FF000'
M12Z20 Constant of X'000FFFFFF'
M16Z16 Constant of X'0000FFFF'
M20Z10 Constant of X'00000FFC'
M20Z12 Constant of X'00000FFF'
M24Z8 Constant of X'000000FF'
M25Z4 Constant of X'0000001C'
M27Z3 Constant of X'0000001C'
M28Z4 Constant of X'0000000F'
M30Z2 Equated to THREE
M8F24 Constant of X'FF000000'
M8Z24 Constant of X'00FFFFFF'
M9Z8 Constant of X'007F8000'

NINE Fullword constant of 9
NXTYPAG Fullword used to indicate next page number available

OC One byte used by EVAL to indicate the operator for the current value
ON Equated to DN
ONE Fullword constant of 1
OP Two-word area which contains the op code of the current statement, left-justified and blank-filled on the right
OPDABS Equated to ZOPD+20. Used to indicate limit for register, shift, length, or immediate data specifications
OW One-byte code which indicates the last operator encountered
OW2 One-byte code which indicates the last operator encountered

PADCB Equated to PADCP
PADCP Fullword pointer to DCB
PALIND One byte used to indicate when listing data set contains at least one line
PAMACS 224-byte area in which index names for user and system libraries and source names for user and system libraries are retained
PAMLC Address of address constants used to close macro library data sets
PAMLCT Fullword that indicates the number of macro libraries available to the assembler
PAPGMS Fullword save area for value of program mask upon entry to the assembler master control
PAPLP Fullword pointer to parameter list supplied by LPC
PASAVE 19-word save area used when calling modules external to the assembler
PC001 Fullword used to indicate the displacement of the location counter value in bits
PDISD One byte used to indicate whether an internal symbol dictionary has been requested
PDISDL One byte used to indicate when ISD listing is required
PDOL One byte used to indicate when an object listing is required
PDPMDL One byte used to indicate when a PMD listing is required
PDSL One byte used to indicate when a source listing is required
PDSTL One byte used to indicate when a symbol table listing is required
PDXL One byte used to indicate when a cross-reference listing is required
PD800 Fullword pointer to working version of USING-REGISTER table
PD801 Equated to PRGSM
PD802 Fullword pointer to ISD chain of USING-REGISTER table pointers
PD803 Fullword pointer to ISD USING-REGISTER table pointer
PD804 Equated to ZUP+20. Fullword used as work area
PEQU Four bytes indicating last "EQU *" dictionary item address
PE001 Fullword pointer to beginning of output text

PE002 Fullword used to indicate the length of the current output text
 PE003 One byte used to indicate to PUTVAL that a complex definition modifier is being processed
 PE004 Fullword used as counter for reference numbers
 PE005 Equated to BASCCS
 PE990 Doubleword work area used for decimal conversion
 PFCALL Location of adcon list used by Phase IV control to call PMD listing processor
 PGSIZE Equated to absolute value of 58
 PLINE Fullword indicating penultimate source line number
 PL008 Fullword pointer to last item linked in literal chain
 PL012 Fullword work area used by POOLIT
 PMDLNG Fullword used to indicate the length of the PMD
 PMDSLCL Fullword pointer to beginning location of program module dictionary
 PMDWLCL Fullword pointer to end of PMD
 PMD Program module dictionary
 PRCTL One byte used to indicate the print options in effect
 PREVCD Fullword used to indicate when a character string continues onto another card
 PRGSM Fullword pointer to previous GSM entry
 PRVGSML Equated to PRGSM
 PRVLOF Fullword pointer to LOF for last statement processed previous to the current statement
 PV001 Fullword which contains the absolute value associated with an expression
 PV002 Equated to PV003+4. Save area for address of RLD string
 PV003 Fullword pointer to end of RLD string
 PV006 Fullword pointer to entry in STACK for last complex term of a complex expression in an address constant
 PV009 Fullword used to indicate the operator of the last complex term processed in a complex expression
 PV013 Fullword used as save area for displacement from the beginning of control section associated with a given symbol
 PV901 Fullword pointer to beginning of reference item chain
 PV902 Fullword pointer to beginning of complex DEF chain
 PV903 Fullword pointer to beginning of external REF chain
 PV904 Fullword pointer to beginning of internal REF chain
 PZ007 Two word save area for pointers to the beginnings of working segments 1 and 2, respectively

RCOL One byte used to indicate rightmost column of field to be sequence checked
 RDBUF 150-byte area into which input lines are read
 RDCNT Fullword used to indicate the length of the line image
 RDLOC Fullword pointer to line image
 RDPL Address of parameter list supplied to GETLINE
 RDSEQ Fullword pointer to previous sequence field
 RDSSW Fullword pointer to source input mode switch
 RD037 Equated to M0Z30
 RD056 Equated to M0Z32
 RD901 Equated to an address in working segment 1. The area beginning at this address is a save area for the previous status
 RD902 Equated to an address in working segment 1. The area beginning at this address is a save area for the current status
 RD903 Ten words used as the STACK area for REED input-mode switches
 RD904 Fullword working storage area used by REED
 RD905 One byte used to indicate whether current line is to be continued
 RD906 Fullword pointer to juncture of concatenated continued lines
 RD907 One byte used by REED to indicate keyboard format
 RD910 One byte used to indicate whether the current line is a continuation of the previous line
 RD911 Fullword save area for limit address for MACRO format
 RD912 One byte used by REED to indicate MACRO format
 RD913 Fullword pointer to beginning of a continued line

RD914 Fullword used to indicate length of a continued line
RD915 Fullword pointer to end of continued line
RD916 Fullword pointer to control bytes for previous source statement
RF010 One byte used to indicate whether statement analyzer was called by the macro reference processor
RF011 Fullword pointer to special entry point in MACREF for the statement analyzer
RF070 Fullword pointer to entry point in SYSXBLD
RF071 Address of parameter list supplied to SYSXBLD
RF072 Address of R-type address constant used in call to SYSXBLD
RF990 19-word save area used when MACREF calls macro retrieval processor
RL001 Fullword used by RESLIT to indicate if expression contained a location counter reference
RL003 Fullword pointer to beginning of literal
RL004 Fullword save area for hash table address
RL005 Two-word save area for key for literal that will be used in a dictionary lookup for the literal
RL008 Fullword save area for length of literal text
RSTADR Fullword pointer to line retrieved from macro library

SALCZ Fullword used to indicate the number of lines used on the page currently being listed
SAPGWA Doubleword work area used when converting characters from one format to another
SAR58 Four-word register save area used by ISD listing processor
SASC Fullword used by CSCAN to indicate whether a constant was or was not specified in the current DS instruction
SAVATT One byte set by DLPM to indicate when symbol being entered has appeared previously on a macro instruction
SAWA Three-word work area used for hex-to-EBC conversion
SCLATR Halfword which indicates scaling attribute of last constant evaluated
SDCTSW One byte used to indicate when symbol is defined within a DSECT
SDLNG Fullword used to indicate length in bytes of ISD output module
SDLOC Fullword pointer to ISD output module
SDSDL Fullword pointer to section number work table
SDSKL Fullword pointer to first symbol sort key -- used by ISD processor
SEQLOC Fullword pointer to current sequence number
SEQSYM Two-word save area that contains the sequence symbol being sought (no blank fill on right)
SETCD Halfword used by SETX to indicate the dictionary item type codes associated with the instruction being processed
SETSS One byte used by SETX to indicate the value of a symbol subscript
SETSUP One-byte code used to control string substitution
SEVCO One byte used to indicate the highest severity code
SEVEN Fullword constant of 7
SIX Fullword constant of 6
SIZE1 Equated to the absolute value 50 -- size of WORK1 GETMAIN
SIZE2 Equated to the absolute value 200 -- size of WORK2 GETMAIN
SIZE3 Equated to the absolute value 20 -- size of each WORK3 GETMAIN
SIZE4 Equated to the absolute value 1 -- size of external name list GETMAIN
SLINE Fullword indicating current source line number
SPACES Eight characters of blanks
SRBUF 132-byte work area in which lines being output by the symbol table listing processor are put into format
SRCODE One byte used by MACREF to indicate the type of statement retrieved from the macro library
SRLIM Fullword pointer to last sort key -- used by CEVSR
SRLNCT Full word used to indicate the number of lines used on the page currently being listed

SRLNG Fullword used to indicate absolute difference between SRLOC and SRLIM
 SRLOC Fullword pointer to first sort key -- used by CEVSR
 SRWK1 Doubleword work area used when converting characters from one format to another
 SRWK2 Three-word work area used when converting characters from one format to another
 SSCATR One byte used by SSCAN to indicate the type of attribute notation
 SSCCUM Fullword used by SSCAN to indicate number of characters in a substring
 SSCFLD One byte used by SSCAN to indicate the type of field being scanned -- name or operand
 SSCL Fullword pointer to location of ampersand
 SSCM Fullword used as count of parenthesis level
 SSCNLI One byte used by SSCAN for a new line indicator
 SSCNLL Fullword pointer to new line location
 SSCNQL Fullword pointer to opening quote of string on new line
 SSCPAK Eight-byte packed work area used by SSCAN to convert value to decimal
 SSCQMF One byte used by SSCAN to indicate when character being scanned is within a quoted string
 SSCSAV Fullword used as register save area by SSCAN
 SSCSSL Fullword used by SSCAN to indicate substring length
 SSCSSP Fullword pointer to closing quote of string
 SSCSSS Fullword pointer to beginning of substring
 SSCSST Fullword pointer to quoted string (or portion thereof)
 SSCSTL Fullword used by SSCAN to indicate length of quoted string
 SSCSYL Fullword used to indicate length of symbol which begins with an ampersand
 SSCT Fullword used by EVAL as a subscript level counter
 SSCUPK 15-byte area used by SSCAN for EBCDIC conversion
 SSLNTH Fullword used to indicate the length of the sequence symbol being sought
 SSM One byte used to indicate whether assembler is in bypass mode
 STACK 16 doublewords used to retain information derived at each parenthesis level, upon the occurrence of a stronger operator, and/or upon the occurrence of a relocatable term
 STAK2 16-word area in which RLD string is saved
 SXBUF 132-byte work area in which lines being output by the source listing processor are put into format
 SXDXSW One byte used to indicate whether a diagnostic is associated with the current statement
 SXHADR Fullword pointer to line that is to be output by CEVSX
 SXLNCT Fullword used to indicate the number of lines used on the page currently being listed
 SXLNG Fullword used to indicate length of the current source line
 SXNXT Fullword pointer to control bytes for a source statement -- used by CEVSX
 SXWK Two doublewords used as work area by CEVSX in converting characters from one format to another

TBLZB Equated to address in working segment 1 which is the beginning of the storage areas used by PARAMAC
 TBLZE Equated to address in working segment 1 which is end of the storage area used by PARAMAC
 TDFLT Four-word table of default attributes
 TEN Fullword constant of 10
 THREE Fullword constant of 3
 TINAME Four bytes that contain the name field of the first TITLE instruction
 TITSW One byte used to indicate whether TITLE instruction has been encountered
 TOS Equated to BSTYPE
 TOTPAG Fullword used to indicate number of pages needed to hold text generated

TRMCT Fullword used by EVAL as a counter for the number of terms in an expression
 TRM1 Character constant of 31
 TWO Fullword constant of 2
 TXTLNG Fullword used to indicate the length of the text module

UDHOLD Equated to ZUSING+16. Used by USE/DROP as save area for register specifications
 UNCAD Fullword save area for uncovered address associated with a symbol
 UNIQUE Equated to beginning of PSECT area
 UP800 Fullword save area for the base register value specified by a USING statement
 UP801 Fullword save area for value that will be placed in first word of a USING-REGISTER table entry
 UP802 Fullword save area for value that will be placed in a USING LOF entry
 UP810 Fullword used as count for external references in USING statements
 UP888 One byte used as code to indicate the type of expression that represents the base register value in a USING instruction
 USETBL Fullword pointer to current USING-REGISTER table

VC Fullword used by EVAL to indicate the value of the current term in an expression
 VMPAGE Fullword pointer to virtual memory page table
 VN Fullword used by EVAL to indicate the value of the next term in an expression

W Eight-doubleword work area used primarily by dictionary routines and expression evaluator
 WCS Fullword used by EVAL to indicate the location counter value associated with the name of the current control section
 WDECS1 Fullword used by CSCAN to indicate the number of leading zeros encountered while scanning a decimal value
 WORK1 Equated to beginning of working segment 1
 WORK2 Equated to beginning of working segment 2
 WORK3 Fullword pointer to the beginning of working segment 3
 WSPCS1 Fullword pointer to STACK relocatable terms
 WSP Fullword pointer to an entry in STACK that represents the first of two relocatable terms that might be paired

XFBUF 132-byte work area in which lines being output by the cross-reference listing processor are put into format
 XFLNCT Fullword used by XREF to indicate the number of lines it has listed on a given page
 XFLNG Fullword used to indicate the number of bytes used in constructing the cross-reference items
 XFLOC Fullword pointer to beginning of cross-reference items
 XFWK1 Two-word work area used for binary-to-hex conversion
 XFWK2 Three-word work area used for binary-to-hex conversion
 XLINE Fullword save area for next line number

ZANOP Five-word save area used by ANOP
 ZBSCAN Seven-word save area used by BASCAN
 ZCATOP Five-word save area used by CATOP
 ZCCW Five-word save area used by CCW
 ZCCWTX Four-word save area used by CCWXTX
 ZCEVAD Seven-word save area used by ADCON
 ZCEVADP 12-word save area used by ADCON
 ZCEVCD Five-word save area used by CSDPR
 ZCEVCY Four-word save area used by COPY
 ZCEVDP 17-word save area used by DCTXT

ZCEVDX Two-word save area used by DIAG
ZCEVEP Four-word save area used by ENDP
ZCEVLS Four-word save area used by LIST
ZCEVMO Nine-word save area used by MOPR
ZCEVPC Three-word save area used by Phase IIB control
ZCEVPP 19-word save area used by Phase IV control
ZCEVPL Three-word save area used by POOLIT
ZCEVPV Three-word save area used by PUTVAL
ZCEVRD Twelve-word save area used by REED
ZCEVSA Six-word register save area used by ISD listing processor SOR
ZCEVSD 14 words used as a register save area by ISD processor
ZCEVSR Seven words used as save area by symbol table listing processor
ZCEVSX Six-word save area used by CEVSX
ZCEVXF Seven-word save area used by XREF
ZCNOP Six-word save area used by CNOP
ZCSCAN Eight-word save area used by constant scan
ZCT Six-word save area used by SECT
ZDCDS Five-word save area used by DC/DS
ZDFSVM Two-word save area used by DEFSYM
ZDOOP Two-word save area used by subroutine internal to EVAL
ZDR Eight-word save area used by DRSET
ZEATT Two-word save area used by subroutine internal to EVAL
ZEBIN Two-word save area used by EBIN
ZECHAR Two-word save area used by ECHAR
ZEDEC Five-word save area used by EDEC
ZEHEX Two-word save area used by EHEX
ZEJECT Two-word save area used by EJECT
ZEND Five-word save area used by END
ZENTRY Five-word save area used by ENTRY
ZEQATE Four-word save area used by EQUATE
ZEQU Eight-word save area used by EQU
ZERO Fullword constant of 0
ZERO12 Constant of Xa00000000a
ZEVAL Five-word save area used by EVAL
ZEXTRN Five-word save area used by EXTRN
ZGETOP Five-word save area used by GETOP
ZGLOC Seven-word save area used by GBLX/LCLX
ZGOIF Five-word save area used by AGO/AIF
ZICTL Five-word save area used by ICTL
ZISEQ Five-word save area used by ISEQ
ZLIT Two-word save area used by subroutine internal to EVAL
ZLITXT Five-word save area used by LITXT
ZLTORG Three-word save area used by LTORG
ZMACRO Five-word save area used by MACRO
ZMCDEF Five-word save area used by MACDEF
ZMCREF Six-word save area used by MACREF
ZMNMXT Five-word save area used by MEND/MEXIT
ZMNOTE Five-word save area used by MNOTE
ZMSCAN Six-word save area used by PARAMAC
ZOPD Six-word save area used by MIP
ZORG Five-word save area used by ORG
ZORGIN Three-word save area used by ORIGIN
ZPARAM Four-word save area used by PARAMAC
ZPRINT Five-word save area used by PRINT
ZPSCAN Two-word save area used by PSCAN
ZPUNCH Four-word save area used by PUNCH
ZRSCON Three-word save area used by RESCON
ZSET Seven-word save area used by SETX
ZSPACE Five-word save area used by SPACE
ZSSCAN Six-word save area used by SSCAN
ZSTAN Five-word save area used by STAN
ZTITLE Four-word save area used by TITLE
ZUP Eight-word save area used by USET
ZUSING Eight-word save area used by USE/DROP
ZUSVAL Five-word save area used by USEVAL
ZVERSN Five-word save area for version identification

APPENDIX E: LIST OF MAJOR TABLES AND WORK AREAS REFERENCED BY ASSEMBLER ROUTINES

Routine Name	Table or Work Area					
	WORK1	WORK2	WORK3	CEVOP	CEVMR	CEVPAS (including ERRIND)
CEVAC	X	X				
CEVAD	X	X	X			X
CEVAN	X					
CEVBS	X		X			X
CEVCC	X	X	X			X
CEVCD	X	X				X
CEVCE	X					
CEVCN	X					X
CEVCP	X		X			X
CEVCS	X	X				X
CEVCT	X	X				X
CEVCW	X					X
CEVCX		X				X
CEVCY	X	X	X			X
CEVDC	X	X				
CEVDF	X	X		X		
CEVDP	X	X				X
CEVDR	X	X				X
CEVDX	X	X				X
CEVEJ	X					
CEVEP	X	X	X			X
CEVEQ	X	X				X
CEVEV	X	X	X			X
CEVEY	X	X				
CEVGB	X		X			X
CEVGC	X		X			X
CEVGD	X		X			X

(Part 1 of 3)

Routine Name	Table or Work Area					CEVPAS (including ERRIND)
	WORK1	WORK2	WORK3	CEVOP	CEVMR	
CEVGH	X		X			X
CEVGL	X	X				X
CEVGN	X	X				X
CEVGO	X		X			X
CEVGP	X		X	X		
CEGVV	X	X				X
CEVGW	X					
CEVIC	X					X
CEVIQ	X					X
CEVKM	X	X				X
CEVLC	X	X				
CEVLG	X	X				
CEVLM	X	X				X
CEVLP	X	X				X
CEVLS	X	X	X			X
CEVLT	X	X				X
CEVMC	X					
CEVMD	X					
CEVMN	X	X				X
CEVMO	X	X	X		X	X
CEVMP	X				X	X
CEVMX	X					X
CEVND	X					
CEVPA	X	X				X
CEVPB	X	X				X
CEVPC	X	X				X
CEVPD	X	X				X
CEVPE	X	X				X
CEVPF	X	X				
CEVPH	X					
CEVPL	X	X				X

(Part 2 of 3)

Routine Name	Table or Work Area					CEVPAS (including (ERRIND
	WORK1	WORK2	WORK3	CEVOP	CEVMR	
CEVPM	X	X				X
CEVPR	X	X				X
CEVPS	X		X			X
CEVPV	X	X				X
CEVQU	X	X				
CEVRD	X	X	X			X
CEVRE						
CEVRF	X	X	X			X
CEVRG	X					
CEVRL	X	X				X
CEVRS	X	X				X
CEVSA	X	X				X
CEVSD	X	X				X
CEVSE	X	X				X
CEVSL	X		X			X
CEVSP	X		X			X
CEVSR	X	X				X
CEVSS	X		X			X
CEVST	X	X	X			X
CEVSX	X	X	X			X
CEVSY	X	X				X
CEVTI	X	X				X
CEVTK	X					X
CEVTM	X	X				
CEVTP	X					
CEVUD	X	X				X
CEVUP	X	X				X
CEVUV	X	X				X
CEVXF	X	X				X
CEVXN	X	X				X

(Part 3 of 3)

INDEX

When more than one page reference is given, the major reference is first.

&SYSECT item 285,287
&SYSLIST item 285,286
&SYSNDX item 285,286-287
&SYSNDX limitation 319
&SYSPSCT item 285,287
&SYSSTYP item 286,287

abnormal termination 7,34-35
absolute value item 261
AC (see assembler control)
acronyms 320-334
ADCON (see address constant processor)
address constant format 293
address constant processor (CEVAD)
 chart CS 238-241
 decision table 102-103
 routine description 109
AGO/AIF instruction scan (CEVGO)
 chart AJ 134
 decision table 41
 routine description 57-58
ANOP instruction scan (CEVAN)
 chart AK 135
 decision table 41
 routine description 58
assembler cleanup (CEVCU)
 chart EC 259
 routine description 313-314
assembler control (CEVAC)
 chart AA 117
 decision table 32
 routine description 33-35
assembler program
 function by instruction type 28-31
 assembler instruction 28-30
 machine instruction 28,29
 macro instruction 28,30
 functional description 9-27
 functions 1,5-7
 instructions 28-31
 interface with LPC 5,7
 limitations 319
 master control 32-35
 organization 2-5
 output 1
 overview 6
 phase control flow 32
 purpose 1
 system environment 1-2
assembler program components (see also individual phases: phase I, phase IIA, phase IIB, phase IIC, phase III, phase IV)
assembler register usage 316
assembler routines, characteristics 8
assembly modules 317-318
assign value to name (CEVEQ)
 chart CB 194-195

decision table 89
routine description 93

BASCAN (see basic source language scan)
basic source language scan (CEVBS)
 chart AI 131-133
 decision table 51
 routine description 56-57
batch mode (see nonconversational mode)
binary self-defining term generator (CEVGB)
 chart BK 173
 decision table 51
 routine description 78
binary text 1
boundary alignment 16
bypass mode 53
CATOP (see string substitution control)
CEVAC (see assembler control)
CEVAD (see address constant processor)
CEVAN (see ANOP instruction scan)
CEVBS (see basic source language scan)
CEVCC (see phase III CCW instruction processor)
CEVCD (see CSD processor)
CEVCE (see SPACE instruction scan)
CEVCN (see CNOP instruction scan)
CEVCP (see string substitution control)
CEVCS (see constant scan)
CEVCT (see control section instruction scan)
CEVCU (see assembler cleanup)
CEVCW (see CCW instruction scan)
CEVCX (see CXD instruction scan)
CEVCY (see COPY instruction processor)
CEVDC (see DC/DS instruction scan)
CEVDF (see macro definition processor)
CEVDP (see phase III constant processor)
CEVDR (see DROP table processor)
CEVDX (see diagnostic message processor)
CEVEJ (see EJECT instruction scan)
CEVEP (see module entry point processor)
CEVEQ (see assign value to name)
CEVEV (see expression evaluator)
CEVEY (see ENTRY instruction scan)
CEVFM (see free VM working storage)
CEVGB (see binary self-defining term generator)
CEVGC (see character self-defining term generator)
CEVGD (see decimal self-defining term generator)
CEVGH (see hexadecimal self-defining term generator)
CEVGL (see global/local symbol instruction scan)
CEVGM (see get VM working storage)
CEVGN (see location counter reset)
CEVGO (see AGO/AIF instruction scan)

CEVGP (see collect and identify operation code)

CEGVV (see obtain relocatable value)

CEVGW (see interface with VISAM PUT or GTWRC macro)

CEVIC (see ICTL instruction scan)

CEVIQ (see ISEQ instruction scan)

CEVKM (see main dictionary lookup)

CEVLG (see LTOrg instruction scan)

CEVLM (see macro name dictionary lookup)

CEVLP (see dictionary lookup and put)

CEVLS (see object program listing)

CEVLT (see phase III literal pool processor)

CEVMC (see MACRO instruction scan)

CEVMD (see program module dictionary listing processor)

CEVMN (see MNOTE instruction scan)

CEVMO (see phase III machine operation processor)

CEVMP (see machine instruction operand scan)

CEVMX (see MEND/MEXIT instruction scan)

CEVND (see END instruction scan)

CEVPA (see phase I control)

CEVPB (see phase IIA control)

CEVPC (see phase IIB control)

CEVPD (see phase IIC control)

CEVPE (see phase III control)

CEVPF (see phase IV control)

CEVPL (see literal pooling processor)

CEVPM (see macro parameter processor)

CEVPR (see PRINT instruction scan)

CEVPS (see parameter item analyzer)

CEVPV (see relocatable output value processor)

CEVQU (see EQU instruction scan)

CEVRD (see obtain next source statement)

CEVRF (see macro reference processor)

CEVRG (see ORG instruction scan)

CEVRL (see literal resolution processor)

CEVRS (see resolve conditional alignment)

CEVSA (see ISD list processor)

CEVSD (see ISD processor)

CEVSE (see SET instruction scan)

CEVSL (see scan for literal operand)

CEVSP (see substitute into operation field)

CEVSR (see symbol table editor)

CEVSS (see string substitution scan)

CEVST (see statement analyzer)

CEVSX (see source listing processor)

CEVSY (see define location symbol)

CEVTI (see TITLE instruction scan)

CEVTK (see lookup temporary dictionary item)

CEVTM (see macro name dictionary put)

CEVTP (see put item in temporary dictionary)

CEVUD (see USING and DROP instruction scan)

CEVUP (see USING table processor)

CEVUV (see compute using register)

CEVXF (see cross-reference listing processor)

CEVXN (see EXTRN instruction operand scan)

CCW instruction scan (CEVCW)
 chart AK 135
 decision table 41
 routine description 58-59

CCWTXT (see phase III CCW instruction processor)

changing storage constants 314

character self-defining term generator (CEVGC)
 chart BL 174
 decision table 51
 routine description 79

CNOP instruction scan (CEVCN)
 chart AL 136
 decision table 41
 routine description 59

collect and identify operation code (CEVGP)
 chart AE 125
 decision table 49
 routine description 54-55

complex value item 263-264

compute using register (CEVUV)
 chart CN 224
 decision table 103
 routine description 106

conditional storage reservations 90

constant item format 293

constant scan (CEVCS)
 chart BG 156-159
 decision table 50
 routine description 71-72

continuation entry 5,34

control section 4-5

control section changes 90

control section dictionary (CSD)
 format 307-311

control section instruction scan (CEVCT)
 chart AM 137
 decision table 46-47
 routine description 59-60

conversational mode

CNOP instruction scan 59

interface with LPC 5

macro instruction processing 83

Phase I functional description 11-12

COPY instruction processor (CEVCY)
 chart AN 138
 decision table 48
 routine description 60-61

cross-reference definition format 113

cross-reference listing 1,27,296-297
 format 296

cross-reference listing processor (CEVXF)
 chart CW 247
 decision table 112
 routine description 112-114

CSCAN (see constant scan)

CSDPR (see CSD processor)

CSD processor (CEVCD)
 chart CU 243-245
 decision table 101
 routine description 110-111

CXD instruction scan (CEVCX)
 chart AL 136
 decision table 42
 routine description 59

data management service routines 1

DC/DS statements, delayed resolution 91

DC/DS instruction scan (CEVDC)

chart AO 139
 decision table 42
 routine description 61
 DCTXT (see phase III constant processor)
 decimal self-defining term generator
 (CEVGD)
 chart BK 173
 decision table 51
 routine description 79
 define location symbol (CEVSY)
 chart BO 177
 decision table 42
 routine description 80
 definition table
 format 309
 DEFSYM (see define location symbol)
 DIAG (see diagnostic message processor)
 diagnostic message processor (CEVDX)
 chart BP 178-179
 decision table 52
 routine description 80-81
 dictionary lookup and put (CEVLP)
 chart BN 176
 decision table 50
 routine description 79-80
 directive code assignments 282
 DLKM (see main dictionary lookup)
 DLKT (see lookup temporary dictionary item)
 DLPM (see dictionary lookup and put)
 documentation modules 317-318
 DPUT (see put item in temporary dictionary)
 DROP table processor (CEVDR)
 chart CG 201
 decision table 96
 routine description 97-98
 DRSET (see DROP table processor)
 DXD item 262-263

early-end entry 7,34
 EBIN (see binary self-defining term
 generator)
 ECHAR (see character self-defining term
 generator)
 EDEC (see decimal self-defining term
 generator)
 EHEX (see hexadecimal self-defining term
 generator)
 EJECT instruction scan (CEVEJ)
 chart AP 140
 decision table 42
 routine description 61
 END instruction scan (CEVND)
 chart AP 140
 decision table 42
 routine description 61
 ENDPR (see module entry point processor)
 ENTRY instruction scan (CEVEY)
 chart AQ 141-142
 decision table 42
 routine description 62
 EQU instruction scan (CEVQU)
 chart AQ 141-142
 decision table 42-43
 routine description 62
 EQUATE (see assign value to name)
 EVAL (see expression evaluator)
 expansion of macro instructions 83-87

Phase IIA functional description 13-16
 processing 3-4
 expression evaluator (CEVEV)
 chart BI 164-171
 decision table 48-49
 routine description 72-77
 external name list 1
 EXTRN instruction operand scan (CEVXN)
 chart AR 143
 decision table 43
 routine description 62-63

free VM working storage (CEVFM)
 chart EB 257-258
 routine description 313
 FREEMAIN macro 312-313

GATEW (see interface with VISAM PUT macro)
 GBLx/LCLx (see global/local symbol
 instruction scan)
 generation data group 1
 get VM working storage (CEVGM)
 chart EA 256
 routine description 312-313
 GETMAIN macro 312-313
 GETOP (see collect and identify operation
 code)
 GETVAL (see obtain relocatable value)
 global/local symbol instruction scan
 (CEVGL)
 chart AS 144
 decision table 43
 routine description 63-64
 global section macro chain (GSM) 280-281
 Part I functional description 9
 Work Area II 7-8
 global section macro entry (GSM) 28,281
 global variable symbol items 269-272
 subscripted items
 arithmetic item 269
 Boolean item 270
 character item 271
 trailer for arithmetic item 269-270
 trailer item 271
 unsubscripted items
 arithmetic item 270
 Boolean item 270-271
 character item 271-272
 global variable symbol pointer items
 269-270,286

hardware requirements 8
 hexadecimal self-defining term generator
 (CEVGH)
 chart BL 174
 decision table 51
 routine description 78-79

ICTL instruction scan (CEVIC)
 chart AT 145
 decision table 44
 routine description 64
 initiation entry 5
 (see also assembler control, CEVAC)

input, LPC and alternate modes 54
 Interface with VISAM PUT or GTWRC macro (CEVGW)
 chart CJ 210
 decision table 103
 routine description 104
 internal symbol dictionary (ISD) 1,300-304
 (see also Figure 5)
 format 303
 limitations 319
 listing 1,298
 listing format 298
 ISD list processor (CEVSA)
 chart DD 255
 decision table 113
 routine description 115
 ISD listing (see internal symbol dictionary)
 ISD processor (CEVSD)
 chart DB 249
 decision table 113
 routine description 114-115
 ISDPR (see ISD processor)
 ISDSA (see ISD list processor)
 ISEQ instruction scan (CEVIQ)
 chart AU 146
 decision table 44
 routine description 64-65

 language processor control (LPC) 1,5-7
 GETLINE function 1
 input 53-54
 interface control flow 7
 PUTDIAG function 1
 LIST (see object program listing)
 literal
 assignment 16
 operands 91
 origin entry 273-274
 origin statements 90
 pooling 16
 literal pooling processor (CEVPL)
 chart CA 193
 decision table 90
 routine description 92-93
 literal resolution processor (CEVRL)
 chart CD 198
 decision table 90
 routine description 94
 LITXT (see phase III literal pooling processor)
 local variable symbol items 268,286
 subscripted
 arithmetic item 289
 Boolean item 290
 character item 290-291
 trailer 289-290
 unsubscripted
 arithmetic item 289
 Boolean item 290
 character item 290-291
 LOCATE (see location counter assignment)
 location counter 4,292
 location counter assignment (CEVLC)
 decision table 89
 routine description 91-92
 location counter reset (CEVGN)
 chart BW 192
 decision table 89
 routine description 92
 logical expressions 75
 logical order file (LOF) 3,272
 (see also Figure 5, Work Area 2)
 alignment specification entry 277
 constant-definition entry 274
 diagnostic message entry 277-278
 END entry 279
 general format 279-280
 literal origin entry 273-274
 machine operation entry 272
 macro instruction entry 273
 MNOTE* entry 278-279
 origin entry 275
 PRINT entry 276
 SET entry 276-277
 TITLE entry 278-279
 USING entry 275-276
 lookup temporary dictionary item (CEVTK)
 chart BQ 180
 decision table 50
 routine description 81
 LTORG instruction scan (CEVLG)
 chart AU 146
 decision table 44
 routine description 65

 MACDEF (see macro definition processor)
 machine instruction directive codes 282
 machine instruction operand scan (CEVMP)
 chart AH 128-130
 decision table 45
 routine description 56
 machine instruction synthesis 5
 machine instructions 29
 machine operations requirements
 table 282-283
 MACLKT (see macro name dictionary look-up)
 MACPUT (see macro name dictionary put)
 MACREF (see macro reference processor)
 MACRO (see MACRO instruction scan)
 macro definition processor (CEVDF)
 chart BF 155
 decision table 44
 routine description 70-71
 MACRO instruction scan (CEVMC)
 chart AV 147
 decision table 45
 routine description 65
 macro instructions 28
 entry 273
 expansion 13-14,83-87
 limitations 319
 processing 3
 macro level dictionary (temporary dictionary) 284-292
 general layout 284-285
 item types
 &SYSECT item 285,287
 &SYSLIST item 285,286
 &SYSNDX item 285,286-287
 &SYSPSCT item 285,287
 &SYSSTYP item 286,287
 global variable symbol pointer item 286,291
 local variable symbol items 268,286

- arithmetic 289-290
- Boolean 290
- character 290-291
- parameter item 286,288
- sequence symbol item 286,289
- macro libraries 1
- macro name dictionary 281
- macro name dictionary lookup (CEVLM)
 - chart BR 181
 - decision table 50
 - routine description 82
- macro name dictionary put (CEVTM)
 - chart BR 181
 - decision table 51
 - routine description 82
- macro parameter processor (CEVPM)
 - chart BU 184-186
 - decision table 85
 - routine description 86-87
- macro reference processor (CEVRF)
 - chart BE 153-154
 - decision table 44-45
 - routine description 70
- macro statement generation 14
- main dictionary 260
 - absolute value term 261
 - basic format 260
 - complex value item 263-264
 - control section item 265-266
 - DXD item 262-263
 - entry trailer item 266
 - external name item 264-265
 - global variable symbol items 269
 - literal item 266-268
 - literal trailer item 267-268
 - local variable symbol items 268-269
 - relocatable value item 261-262
 - sequence symbol item 272
 - transitive item 268
- main dictionary lookup (CEVKM)
 - chart BS 182
 - decision table 52
 - routine description 82
- MEND/MEXIT instruction scan (CEVMX)
 - chart AV 147
 - decision table 45
 - routine description 65-66
- MIP (see machine instruction operand scan)
- MNOTE instruction scan (CEVMN)
 - chart AW 148
 - decision table 45
 - routine description 66
- module entry point processor (CEVEP)
 - chart CK 211
 - decision table 101
 - routine description 104
- MOPR (see phase III machine operation processor)

- nonconversational mode
 - CNOP instruction scan 59
 - Phase I functional description 11
- normal mode 52
- normal statements 91

- object program listing 1,298-300
 - format for constants 301
 - format for machine and assembler instruction 302
- object program listing (CEVLS)
 - chart CO 225-230
 - decision table 103
 - routine description 106-107
- object text generation 16-20
- obtain next source statement (CEVRD)
 - chart AD 123-124
 - decision table 46
 - routine description 53-54
- obtain relocatable value (CEGVV)
 - chart CM 223
 - decision table 103
 - routine description 105-106
- operation code table 11,281-282
- operators, hierarchy of 76
- ORG instruction scan (CEVRG)
 - chart BA 149
 - decision table 45
 - routine description 66-67
- ORG statements 90
- ORIGIN (see location counter reset)
- output options 1,26,112-115
- overflow diagnosis 314

- page usage (see also Figure 5)
 - recording 91
 - tables 7,18
- PARAMAC (see macro parameter processor)
 - parameter item 286,288
 - macro parameter processor (CEVPM) 86-87
- parameter item analyzer (CEVPS)
 - chart BJ 172
 - decision table 51
 - routine description 77-78
- parentheses
 - hierarchy 76
 - interpretation of 74
 - scanning for 73,75
- phase I 36-82
 - (see also phase I control)
 - assembly modules 317-318
 - communication with external routines 36
 - documentation modules 317-318
 - functional description 9-13
 - overview 10
 - routine relationships 37-38
- phase I control (CEVPA)
 - chart AB 118
 - decision table 39
 - routine description 36
- phase IIA 83-87
 - (see also phase IIA control)
 - assembly modules 317-318
 - documentation modules 318
 - functional description 13-16
 - overview 15
 - routine relationships 84
- phase IIA control (CEVPB)
 - chart BT 183
 - decision table 84
 - routine description 83-86
- phase IIB 89-94
 - (see also phase IIB control)

assembly module 318
documentation modules 318
functional description 16-20
overview 17
routine relationships 88
phase IIB control (CEVPC)
chart BV 187-191
decision table 89
routine description 88-91
phase IIC 95-98
(see also phase IIC control)
assembly module 318
documentation modules 318
functional description 20-22
overview 21
routine relationships 95
phase IIC control (CEVPD)
chart CE 199
decision table 95-96
routine description 96-97
phase III 99-111
(see also phase III control)
assembly modules 318
documentation modules 318
functional description 22-26
overview 22
routine relationships 100
phase III CCW instruction processor
(CEVCC)
chart CP 231-232
decision table 101
routine description 107
phase III constant processor (CEVDP)
chart CR 235-237
decision table 102
routine description 108
phase III control (CEVPE)
chart CH 202-207
decision table 100-101
routine description 99
phase III literal pooling processor
(CEVLT)
chart CT 242
decision table 101
routine description 109-110
phase III machine operation processor
(CEVMO)
chart CL 212-222
decision table 101-102
routine description 104-105
phase IV 112-115
(see also phase IV control)
assembly module 318
documentation modules 318
functional description 26-27
overview 26
routine relationships 112
phase IV control (CEVPF)
chart CV 246
decision table 113
routine description 112
PMD listing (see program module dictionary)
PMDLS (see program module dictionary
listing processor)
POOLIT (see literal pooling processor)
post processing 5
PRINT (see PRINT instruction operand scan)
print control 23
PRINT instruction operand scan (CEVPR)
chart BA 149
decision table 46
routine description 67
program module dictionary (PMD) 1,304-307
format 305-306
limitations 319
listing 1,26,297-298
listing format 299
Work Area 3 8
program module dictionary listing processor
(CEVMD)
chart DC 250-254
decision table 113
routine description 115
program reordering 4
PSCAN (see parameter item analyzer)
pseudo-dictionary item for current location
counter 292-293
PUNCH instruction scan (CEVPH)
decision table 46
routine description 67-68
push-down-stack logic 4
put item in temporary dictionary (CEVTP)
chart BQ 180
decision table 50
routine description 81
PUTVAL (see relocatable output value
processor)

REED (see obtain next source statement)
reference item format 114
reference table format 309-310
register usage 313
relocatable expression reduction 75
relocatable output value processor (CEVPV)
chart CQ 233-234
decision table 103
routine description 107-108
relocation dictionary (RLD) format 310-311
REPRO instruction scan (CEVRE)
decision table 46
routine description 67-68
RESCON (see resolve conditional alignment)
RESLIT (see literal resolution)
resolve conditional alignment (CEVRS)
chart CC 196-197
decision table 90
routine description 93-94

scan for literal operand (CEVSL)
chart BM 175
decision table 47
routine description 79
scanning techniques 73
SECT (see control section instruction scan)
section name table 114,300
sequence symbol item 272,289
SET instruction scan (CEVSE)
chart BB 150
decision table 47
routine description 68
SETA (see SET instruction scan)
SETB (see SET instruction scan)
SETC (see SET instruction scan)
SETX (see SET instruction scan)

SLIT (see scan for literal operand)
 SLLS (see source listing processor)
 source line storage control 292
 source listing processor (CEVSX)
 chart CI 208-209
 decision table 102
 routine description 104
 source program
 correction facility 54
 listing 1,99,294
 listing format 295
 source statement limitations 319
 SPACE instruction scan (CEVCE)
 chart BC 151
 decision table 47
 routine description 68-69
 SSCAN (see string substitution scan)
 STAN (see statement analyzer)
 statement analyzer (CEVST)
 chart AC 119-122
 decision table 39-40
 routine description 36,54
 statement analyzer, modes of operation
 bypass 54
 normal 53
 statement length, maximum 319
 STED (see symbol table editor)
 storage request constants
 changing 314
 table of 315
 string substitution control (CEVCP)
 chart AG 127
 decision table 41
 routine description 55-56
 string substitution scan (CEVSS)
 chart BH 160-163
 decision table 51-52
 routine description 72
 SUBOP (see substitute into operation code
 field)
 subscripted items
 LCLA 289
 LCLB 290
 LCLC 290-291
 subscripted values, retrieval of 76
 substitute into operation code field
 (CEVSP)
 chart AF 126
 decision table 51
 routine description 55
 symbol definition 16-20
 symbol entries 115
 symbol table 300
 dictionary 9
 listing 1,26,295-296
 listing format 296
 symbol table editor (CEVSR)
 chart DA 248
 decision table 113
 routine description 114
 syntax analysis 3
 system device (SYSOUT) 5

 tables and work areas 335-337
 temporary dictionary (see macro level
 dictionary)
 termination of processing 91
 abnormal 7,34-35
 text locator table 80
 TITLE instruction scan (CEVTI)
 chart BC 151
 decision table 47-48
 routine description 69

 unsubscripted items
 LCLA 289
 LCLB 290
 LCLC 290-291
 USE/DROP (see USING and DROP instruction
 scan)
 user virtual storage 7
 USET (see USING table processor)
 USEVAL (see compute using-register)
 USING and DROP instruction scan (CEVUD)
 chart BD 152
 decision table 48
 routine description 69-70
 USING table processor (CEVUP)
 chart CF 200
 decision table 96
 routine description 97
 using-register tables 283-284

 virtual memory management 312-315
 virtual memory management table
 (VMTABLE) 293-294
 virtual memory page table (VMPT)
 format 311
 virtual storage requirements 7-8,312-315
 VMCLEAN (see assembler cleanup)
 VMFREE (see free VM working storage)
 VMGET (see get VM working storage)
 VMPT (see virtual memory page table)
 VMTABLE (see virtual memory management
 table)

 working storage areas 7-8

 XREF (see cross reference listing
 processor)

IBM

**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]**